UNIVERSIDAD POLITÉCNICA DE CARTAGENA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INDUSTRIAL

DEPARTAMENTO DE TECNOLOGIA ELECTRÓNICA

PROYECTO FIN DE CARRERA

INGENIERO EN AUTOMÁTICA Y ELECTRÓNICA INDUSTRIAL

"CONTROL INALÁMBRICO BASADO EN REDES INALÁMBRICAS DE SENSORES MEDIANTE MÓDULOS XBEE"

DIRECTORES: D. Fulgencio Soto Valles
D. Juan A. López Riquelme

AUTOR: José Navarro Muñoz

Cartagena, Julio de 2013

AGRADECIMIENTOS

En primer lugar he de agradecer el gran apoyo recibido por toda mi familia

A mis padres por sus sacrificios y haberme inculcado la curiosidad por las cosas desde niño

A mi hermano por su insistencia y por saber ayudarme en los momentos más difíciles

A María Isabel por toda su comprensión demostrada y el ánimo que me ha infundido

A mis tíos y tías por aportarme un punto de vista diferente

A mis directores Juan Antonio y Pencho por su gran paciencia y ayuda en la elaboración de este proyecto

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE ILUSTRACIONES

Capítulo 1

Introducción

1.1 Introducción

En la planta de exprimido de la empresa Confridul localizada en Jumilla, debido al crecimiento que ha experimentado estos últimos años, el tamaño actual de la planta tratadora de aguas sucias provenientes de la limpieza de la fruta no es suficiente y su actual ubicación no permite una ampliación.

Para dar solución a este problema, se han adquirido unos terrenos fuera de las actuales instalaciones, donde se ha instalado el nuevo centro de tratado de las aguas residuales. Este centro de tratado consiste en dos reactores biológicos, formados a su vez por dos tanques abiertos de gran capacidad. En el primero se airea el agua sucia para aumentar la cantidad de oxígeno en la disolución, además de que las bacterias aerobias puedan reproducirse con facilidad y consumir la materia orgánica. En el segundo, es donde se almacena el agua durante un tiempo, típicamente 4 horas, para que decante toda la materia sólida y que las bacterias anaeróbicas que están presentes en la capa de fango del fondo del tanque terminen el proceso con la destrucción de la materia orgánica todavía presente, quedando el agua así apta para su reutilización. Esta agua ya tratada se va utilizando según las necesidades y sólo es necesario extraerla por la salida de servicio, que está colocada a dos tercios de la altura del tanque, con objeto de extraer sólo el agua ya decantada y que no se pierda la capa de fangos donde viven las bacterias anaeróbicas.

El nuevo proceso de aireado requiere que el nivel del agua en el primer tanque sea siempre constante, ya que la aireación no es regulable. Por ello, si el nivel de este tanque es

demasiado bajo, los aireadores hacen que las aguas salpiquen y se agiten demasiado. Esto produce inestabilidades en el tanque, al igual que si el nivel es demasiado elevado el agua rebosa.

Por tanto, resulta de vital importancia controlar el nivel de aportación al primer tanque, que ha de mantener su nivel y que oscila según las necesidades de consumo de agua tratada del tanque 2. El llenado se realiza mediante la bomba de las instalaciones viejas, colocada a la salida del tanque reutilizado como recolector de las aguas sucias de la planta. La distancia entre la planta de tratado antigua y las nuevas instalaciones es de 800 metros. Puesto que, entre ambas instalaciones pasa una carretera, instalar la infraestructura requerida por un sistema de control tradicional supondría un gran coste. Por este motivo, se ha decidido la instalación de la medición de caudal del tanque 1 de la nueva instalación y el accionamiento de la bomba de la plataforma antigua usando redes inalámbricas de sensores. Esto permite a un operador situado en las oficinas de la antigua instalación, supervisar y controlar el tratamiento de las aguas.

Una red inalámbrica de sensores se basa en interconectar pequeños dispositivos inalámbricos de bajo consumo, con el objetivo de formar una red que sea capaz de recopilar información de cada uno de los sensores conectados, hasta un sistema central, que se encargará de reaccionar y enviar las acciones correctoras, a partir de la información recibida a través de la red, hasta los dispositivos con posibilidad de actuar con el sistema.

Las redes inalámbricas de sensores se han convertido en un campo de estudio que se encuentra en continuo crecimiento, ya que en los últimos tiempos han surgido una serie de dispositivos que integran un procesador, una pequeña memoria, sensores, y que están dotados de comunicación inalámbrica. A estos dispositivos inalámbricos se les imponen unas restricciones de consumo severas. El motivo de la imposición de estas restricciones es la necesidad de que los dispositivos sean capaces de operar, durante periodos largos de tiempo alimentados con baterías, que puedan ser recargadas mediante energía solar.

El ámbito de aplicación de este tipo de sistemas es muy amplio: monitorización de entornos naturales, aplicaciones para defensa, aplicaciones médicas en observación de pacientes, monitorización en el ámbito industrial, etc. En este caso de estudio, la tecnología de las redes inalámbricas de sensores resulta muy útil, ya que permite supervisar y controlar el estado de la instalación de aguas residuales de forma remota y eficiente, sin que sea necesario realizar una costosa y complicada infraestructura, que sería necesaria en una solución cableada.

1.2 Objetivos

El principal objetivo de este proyecto fin de carrera es proponer una solución óptima y de bajo coste, usando la tecnología de las redes inalámbricas de sensores, para poder controlar subprocesos de instalación industrial de forma remota, sin perder fiabilidad y consiguiendo un ahorro sustancial respecto a realizar estas tareas con módulos cableados. Para conseguir este objetivo será necesario llevar a cabo los siguientes sub-objetivos:

- Revisar el estado del arte de las redes de sensores inalámbricas.
- Estudio del kit de desarrollo XBee 802.15.4 *Starter Development Kit.*
- Realización de pruebas de los módulos XBee mediante su software de configuración.
- Realización de pruebas de los módulos XBee mediante su interfaz API.
- Realización de una aplicación de PC para comunicar con los XBee.
- Desarrollo placas específicas para los XBee, donde se adaptan las señales de tipo industrial a las requeridas por los módulos.
- Desarrollo de una aplicación específica de PC, para tareas de supervisión y control en instalación industrial, mediante una red con XBee.

1.3 Fases del Proyecto

En la primera parte del proyecto se plantearan los módulos basados en tecnología 802.15.4 de diferentes fabricantes. Se elige el módulo XBee de la empresa "Digi" y se experimenta sobre las capacidades del módulo, mediante el uso de su kit de desarrollo para conocer a fondo sus posibilidades y limitaciones.

Una vez conocido el funcionamiento de los módulos XBee, se resolverá su interconexión con el sistema central, que en este caso es un PC. Dicho elemento del sistema actuará de puesto de control. Para ello, se desarrolla el software necesario para el control y monitorización. Mediante la realización de un caso de estudio, se plantea resolver el problema de comunicar dos partes de una instalación industrial, que debido a su crecimiento han quedado separadas. Debido al gran coste que supondría el cableado convencional, se plantea el uso de esta solución inalámbrica. En el caso de estudio se mostrará el proceso a seguir en la resolución de problemas reales con esta tecnología.

Para llevar a cabo las tareas y objetivos descritos se establecen las siguientes fases del proyecto:

- Estudio de las redes de sensores inalámbricos y su tecnología.
- Elección de un módulo de radio comercial.
- Estudio del kit de desarrollo elegido, hardware y software.
- Realización de pruebas de sus funcionalidades.
- Realización de software para PC de comunicaciones con los XBee.
- Desarrollo del hardware para usar los XBee con señales industriales.
- Desarrollo del software para PC específico para la aplicación de monitorización y control industrial.
- Pruebas del sistema usando un sensor industrial y un variador de frecuencia.
- Realización de la memoria de documentación del proyecto.

1.4 Desarrollo de la Memoria

1.4.1 Capítulo 1 - Introducción

El Capítulo 1 presenta la motivación de este trabajo, además de enumerar los objetivos del mismo y describir la memoria presentada.

1.4.2 Capítulo 2. Estado del Arte

En el capítulo 2 se desarrolla un estudio del estándar de comunicación más ampliamente usado en las redes de sensores, el IEEE 802.15.4, sobre el que trabajan los módulos XBee usados en este estudio, así como los principales fabricantes de módulos con chips de radio que usan este estándar.

1.4.3 Capítulo 3. Descripción del Kit de Desarrollo XBee

En el tercer capítulo se describe en profundidad el módulo XBee de "Digi" elegido para el desarrollo de este proyecto, así como los programas software para su manejo y programación.

1.4.4 Capítulo 4. Modos de Operación del XBee

Se especifican las diferentes posibilidades de trabajo de los módulos XBee y se documentan las pruebas realizadas con estos dispositivos. Estas pruebas se han realizado para comprobar su funcionalidad. También se explica el diseño de la primera aplicación software desarrollada para la realización de las pruebas de las funcionalidades del API (Interfaz de programación de aplicaciones) de los módulos inalámbricos XBee.

1.4.5 Capítulo 5. Caso de Estudio

Se terminará el estudio con el planteamiento de un caso práctico, en el que se plantea el uso de los XBee para resolver un problema de control automático con sensores y actuadores distanciados entre sí. En este capítulo se describe la aplicación software desarrollada en este trabajo, y que permite comunicarse con los módulos XBee en el modo API. Esta aplicación también permite realizar el diseño de una interfaz de usuario totalmente personalizable en el PC de control y monitorización.

La aplicación se ha desarrollado usando los principios de programación del uso de patrones, en concreto se ha desarrollado bajo el patrón de software Modelo-Vista-Controlador (MVC), que propone una arquitectura que separa claramente la parte comportamental, de la comunicación de los módulos, con la parte visual que se le muestra al usuario final. Esto permite que con la modificación de solo la vista se pueda variar la aplicación para otros problemas en los que se use una topología de red similar.

1.4.6 Capítulo 6. Conclusiones y Trabajos Futuros

En este capítulo se extraen las principales ideas que se han de tener en cuenta a la hora de trabajar con esta tecnología y se plantean usos en posibles proyectos futuros, basados en los módulos bajo el estándar IEEE 802.15.4.

1.4.7 Anexo I -Descripción de Parámetros del Variador de Frecuencia V1000

En este anexo se describen los parámetros del variador de frecuencia usado en este trabajo. Específicamente, el variador de frecuencia se utiliza para controlar la velocidad de la bomba mediante las tramas recibidas por el módulo XBee con el rol de actuador. Este nodo de la red está conectado físicamente con el variador de frecuencia. En relación al modelo y fabricante, se trata del V1000 del fabricante Omron.

1.4.8 Anexo II – Descripción del PID Digital

En este anexo se describe el proceso que se ha llevado a cabo para obtener la ecuación de un PID digital. Un PID es el mecanismo de control más ampliamente utilizado para regular procesos en el sector industrial. La ecuación analógica del mismo es muy conocida y puede ser implementada mediante componentes de electrónica analógica, siendo muy habitual la implementación basada en amplificadores operacionales. Sin embargo, cuando se pretende utilizar dicha estrategia de control sobre un sistema basado en un micro-procesador, es necesario obtener una versión digital del mismo.

1.4.9 Anexo III – Descripción de la Aplicación de Control

En este anexo se describen las diferentes clases que se han implementado para obtener la aplicación de control que se ejecuta en el PC. Concretamente, las clases se han implementado en lenguaje Java y siguiendo el patrón de diseño denominado Modelo Vista Controlador.

1.4.10 Anexo IV – Configuración de los Módulos XBee

En este anexo se detallan todos los comandos AT de configuración de los tres módulos XBee usados en el proyecto: el módulo estación base, el módulo actuador y el módulo sensor. Además, al final del documento, se ha incluido una tabla resumen con todos los comandos AT que se pueden configurar de los módulos XBee.

Capítulo 2

Estado del Arte

2.1 Introducción

En este capítulo se va a hacer una revisión de las redes inalámbricas de sensores, las cuales constan de un conjunto de elementos con capacidades sensoriales que transmiten su información inalámbricamente al resto de elementos bajo el estándar IEEE 802.15.4. El estándar IEEE 802.15.4 está especialmente diseñado para trabajar con bajas tasas de transferencia y bajo consumo de sus elementos lo que es especialmente importante para estas redes. Las WSN están formadas por motes, los cuales constan de un microprocesador, un módulo de radio con su correspondiente antena, además de una batería como fuente de energía.

Para familiarizarse con los motes existen en el mercado los kits de desarrollo, los cuales están formados por pequeñas placas electrónicas con lo necesario para empezar con el diseño y la configuración de los motes, estos circuitos incorporan métodos de conexión al PC a través del cual se programan, conectores para hacer accesibles los pines, el sistema de adaptación de la alimentación, incorporando además la mayoría de estos kits, LED's y pulsadores con los que interactuar con el mote. Estos kits permiten el posterior desarrollo de aplicaciones basadas en ellos.

2.2 Redes Inalámbricas de Sensores

Una red inalámbrica de sensores (WSN, *Wireless Sensor Network*) [1] es un conjunto de dispositivos autónomos distribuidos espacialmente utilizando sensores para supervisar conjuntamente parámetros físicos o condiciones ambientales, como temperatura, sonido, vibración, presión, movimiento o contaminantes, en distintos lugares. El desarrollo de las redes inalámbricas de sensores estuvo inicialmente motivado por las aplicaciones militares, como la vigilancia del campo de batalla. Sin embargo, las redes inalámbricas de sensores en la actualidad se utilizan en muchas áreas de aplicación civil, incluidos el control del tráfico, atención sanitaria, domótica y otros muchos campos, como se muestra en la Ilustración 2.1. El diseño eficiente e implementación de las redes inalámbricas de sensores se ha convertido en un área de investigación emergente en los últimos años, debido al gran potencial de estas redes para cubrir grandes áreas a bajo costo.

Ilustración 2.1: Aplicaciones de las WSN

El tamaño de un nodo sensor también llamado Mote, puede variar, al igual que el coste, dependiendo del tamaño de la red y de la complejidad computacional requerida por los distintos nodos sensores. El tamaño y la limitación del consumo en los nodos sensores son los causantes de las limitaciones de recursos como la potencia de transmisión (<50mw), factor decisivo en el despliegue de los nodos sensores, la memoria (<1Mbyte), la velocidad de cálculo (<500 MHz y el ancho de banda (250kbps). Cuando la WSN es colocada en ciertas zonas, como en un bosque o bajo el agua, no es posible mantenerlas bajo supervisión continua, por lo tanto, deben tener la suficiente energía para maximizar su tiempo de vida. Las especificaciones pueden dividirse principalmente en dos categorías: software y hardware. El Software debe incluir algoritmos de baja complejidad y bajo tiempo de ejecución, adaptaciones para cambios en el medio inalámbrico, un algoritmo eficiente de enrutamiento y protocolos de acceso al medio, todos ellos que requieran un

bajo consumo de energía. El hardware incluye la limitación de la disponibilidad de circuitos integrados o IC, el tamaño del mote, bajo consumo de energía, bajo coste de producción, la capacidad de adaptación al medio ambiente, etc.

Los sensores inalámbricos pueden utilizarse para recoger diferentes tipos de información dependiendo de los diferentes sensores que se coloquen en la misma red. Por lo tanto, se pueden tener diferentes datos, al mismo tiempo, es decir acústicos, sísmicos, ambientales, etc. transmitidos a través de la red inalámbrica de sensores a la estación base, la cuál es un nodo especial de la red, formado por un procesador un módulo de radio, una antena y un sistema de conexión a un PC, este nodo se encarga de recoger lo datos enviados por el resto de nodos y enviarlos a un PC o a otra red a través de una pasarela, donde se analizarán y tratarán estos datos. Finalmente se puede decir que los sensores inalámbricos son un paso de gigante hacia la computación proactiva, un paradigma donde las computadoras teniendo la información recogida del medio por los sensores, son capaces de tomar decisiones para prevenir situaciones no deseadas desde los primeros síntomas detectados.

De los elementos que componen una WSN, se pueden ver los más relevantes en la Ilustración 2.2, caben destacar los nodos sensores, la estación base o sumidero, y la red inalámbrica propiamente dicha.

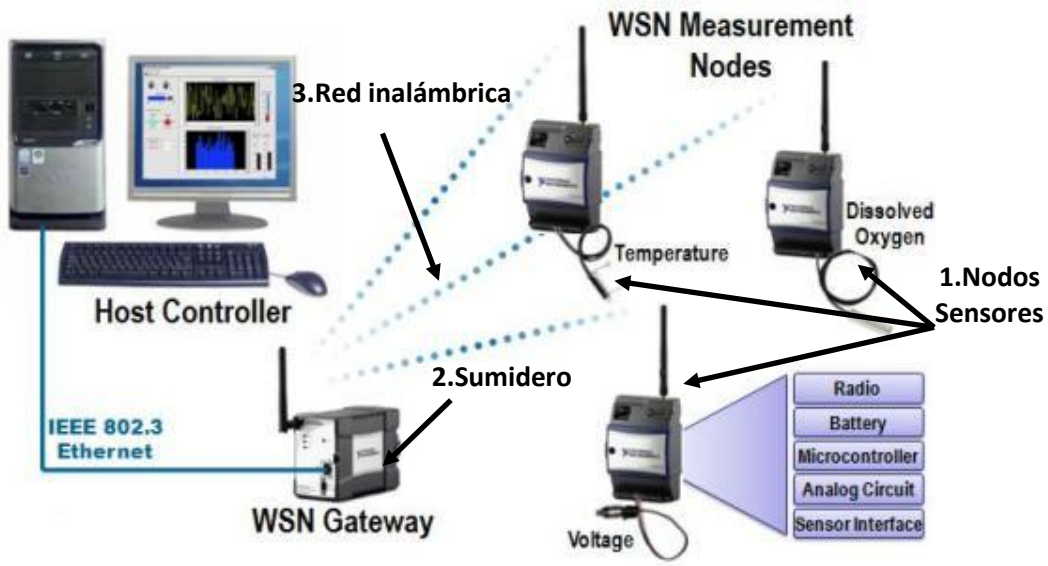


Ilustración 2.2 : Elementos de las WSN

2.3 Estándar IEEE 802.15.4

Este estándar se usa en las WSN, porque permite comunicaciones fiables y consigue bajos consumos en los nodos sensores, con la contrapartida de tener bajas tasas de transferencia de datos.

2.3.1 Introducción

El IEEE 802.15.4 [2] es un estándar que define el nivel físico y el control de acceso al medio de redes inalámbricas de área personal con tasas bajas de transmisión de datos (LR-WPAN, *Low Rate - Wireless Personal Area Network*).

Una LR-WPAN es una red de comunicación simple, de bajo costo que permite conectividad entre aplicaciones con requerimientos de potencia limitada y un rendimiento flexible. Los requerimientos de estas redes son, ser fáciles de instalar, tener fiabilidad en los datos enviados, disponer de un pequeño rango de alcance, tener bajo costo y un consumo muy moderado para alargar la vida de las baterías, todo esto junto a un protocolo simple y flexible.

Las principales características de estas redes son las siguientes:

- Ratio de transferencia de 250Kbps, 100Kbps, 40Kbps y 20Kbps.
- Topología de red en Estrella o comunicación *peer-2-peer*. (módulo a módulo)
  (Más adelante se detalla esta topología de red y su funcionamiento)
- Direccionamiento de 16 o 64 bits.
- Asignación opcional de espacio de tiempo garantizado para la transmisión(GTS).
- Acceso al medio mediante CSMA-CA, [3] (*Carrier Sense Multiple Access with collision Avoidance* , acceso múltiple por detección de portadora con evasión de colisión).
- Protocolo completo de *token* de confirmación (ack), para conseguir transferencias fiables.
- Bajo consumo de potencia.
- Detección de energía (ED, *Energy Detection*)
- Indicador de la calidad de la conexión (LQI, *Link Quality Indication*)
- 16 canales en la banda de 2'4GHz, 30 canales en la banda de 915MHz y 3 canales en la de 868MHz.

En redes bajo el estándar IEEE 802.15.4 se pueden encontrar dos tipos diferentes de participantes, el FFD (*Full Function Device*- Dispositivo de juego completo de funciones) y RFD, (*Reduced Function Device* – Dispositivos de juego reducido de funciones). Los FFD pueden seguir cualquiera de estos tres roles, Coordinador, dispositivo final y coordinador de red de área personal (PAN – *Personal Area Network*). Un FFD puede dialogar con cualquier otro FFD o con otros RFD, mientras que estos últimos solo pueden dialogar con FFD, esto es, los RFD no pueden dialogar con otros RFD como ellos. Los dispositivos RFD están pensados para realizar tareas simples y que no requieran enviar grandes tramas

de datos ni estar asociados con más de un FFD a la vez. Consecuentemente implementar RFD requiere menos recursos y menor capacidad que un FFD.

Estas redes no tienen un área de cobertura totalmente definida porque los mecanismos de propagación son dinámicos, por lo que un pequeño cambio en la alineación u orientación de los dispositivos puede hacer que varíe de manera muy significativa la calidad de la comunicación.

2.3.2 Topología de Red

Las redes que se construyen dentro del estándar IEEE 802.15.4 deben de autorganizarse y automantenerse para que de esta forma se reduzcan los costes totales y se facilite su uso.

Dependiendo de los requerimientos de la aplicación un red bajo el estándar IEEE 802.15.4 puede operar en cualquiera de estas configuraciones: red en estrella y *peer-to-peer*.

En la configuración en estrella, es necesario un coordinador de PAN hacia el cual los demás módulos dirigen su comunicación, y que es el encargado de iniciar, terminar o redirigir la comunicación a través de la red. Aparte de esto este nodo realiza la propia función que tenga asignada. Por ejemplo, lectura de sensores, actuar como nodo central de recogida de datos, etc. Es posible ver gráficamente como se distribuirían los módulos en estas redes en la Ilustración 2.3.

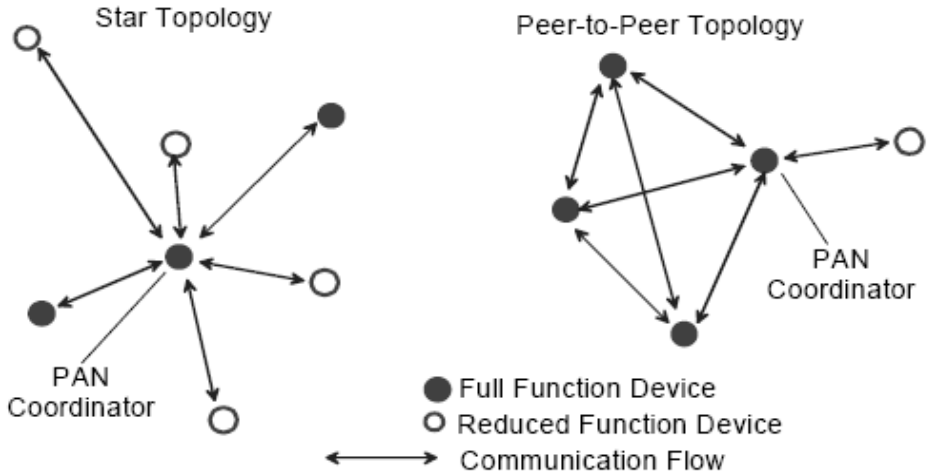


Ilustración 2.3 Topología en estrella y red *peer-to-peer*

En la topología de red *peer-to-peer*, también existe un coordinador de red, que se comunica con todos los módulos, pero además cada módulo puede comunicarse con otros de la misma red que estén a su alcance. Con esta topología de red es posible crear estructuras más complejas como redes enmalladas, se muestra un ejemplo de estas redes en la Ilustración 2.3. Estas redes pueden ser ad-hoc, es decir redes sin infraestructura física

preestablecida, ni necesidad de un control central que coordine su actividad, además pueden tener la propiedad de ser autorganizativas con restauración de caminos, sus sensores pueden trabajar como emisores o receptores y pueden establecer caminos de comunicación entre nodos sin visibilidad directa y modificar estos caminos si alguno de los nodos del encaminamiento falla, etc. pero estas funciones han de ser establecidas en una capa superior, la cual no forma parte de este estándar.

2.3.3 Arquitectura

La definición del estándar está dividida en dos capas, que están basadas en la estructura OSI (*Open Systems Interconnection*- Modelo de referencia de interconexión de sistemas abiertos) [4], concretamente, el estándar IEEE 802.15.4 define la capa de nivel físico, la cual contiene la definición de los métodos de control de bajo nivel de la radio y la capa de nivel de enlace de datos (MAC- *Medium Access Contention*)[5], desde donde se da acceso a la capa física para todos los tipos de transmisiones. En la Ilustración 2.4 se observan las diferentes capas que se definen en la estructura OSI y su correspondencia en el estándar IEEE 802.15.4 así como una equivalencia del resto de las capas implementadas mediante uno de los estándares de nivel superior al IEEE 802.15.4, el denominado ZigBee [6] ,ampliamente usado para este tipo de redes.

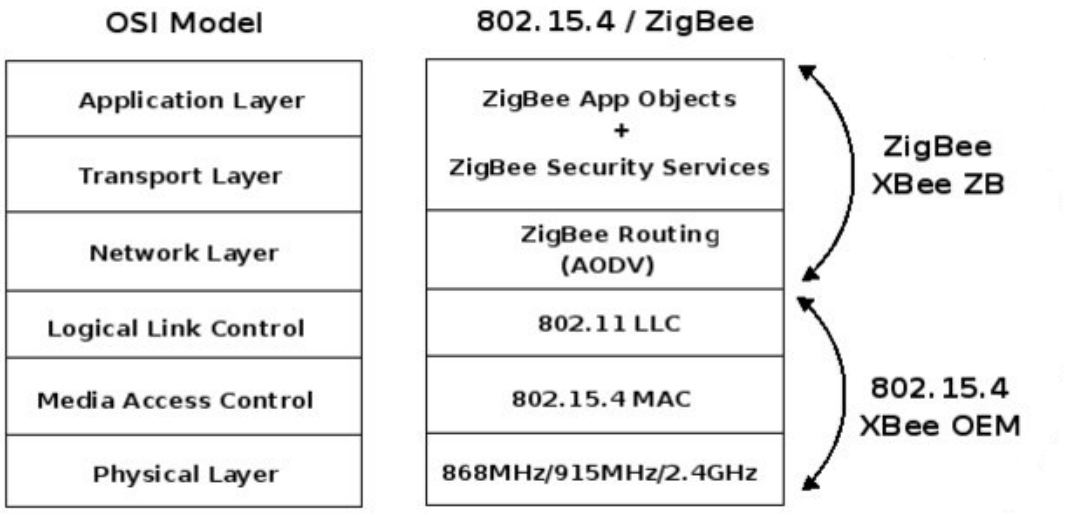


Ilustración 2.4 Integración de capas 802.15.4 en el modelo OSI

Subcapa física

Las funciones de esta capa son: la activación y desactivación de la radio, la selección del canal, la estimación de la calidad de la conexión, la medición de la energía empleada en la comunicación, la detección del canal con menos interferencias (CCA, *Clear Channel*

Assessment), el transmitir y recibir los paquetes a través de la radiofrecuencia, usando para ello tres de las bandas libres ISM (*Industrial, Scientific and Medical*), las cuales están recogidas en la Ilustración 2.5.

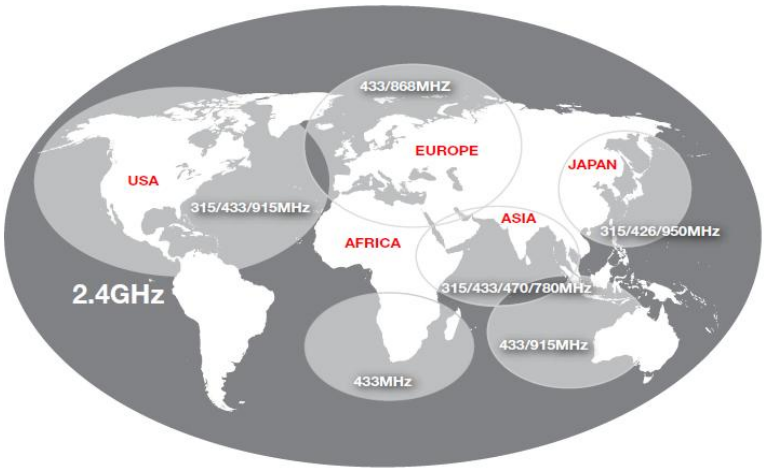


Ilustración 2.5 : Bandas libres ISM en el Mundo

De entre ellas, las frecuencias usadas por la capa física del estándar IEEE 802.15.4 y sus regiones de uso permitido son las siguientes:

- 868.0 - 868.6 MHz (Europa)
- 902 - 928 MHz (North América)
- 2400 – 2483.5 MHz (Internacional)

En la banda 868MHz soporta un canal, en la de 915MHz soporta 10 canales y en la de 2.4GHz soporta 16 canales. En todas ellas el acceso al medio es mediante DSSS (Direct *Sequence Spread Spectrum*), este es un método de modulación de la señal, eficiente en consumo y que consigue gran inmunidad al ruido, con el inconveniente de usar más ancho de banda que otros métodos.

Como técnica de modulación en la banda de 2.4GHz se emplea O-QPSK (*Offset Quadrature Phase Shift*), en este método se consigue aumentar la cantidad de datos a transmitir con el mismo ancho de banda a costa de reducir su inmunidad contra el ruido. En las bandas de 915MHz y 868MHz se emplea la técnica de modulación de BPSK (*Binary Phase Shift Keying*).

En la revisión de 2006 del estándar IEEE802.11.4, se añadió la técnica de modulación ASK (*Amplitude Shift Keying*), con la que se obtienen tasas de transferencia más altas, hasta los 250kb/s para las bandas de 868 Mhz y 915 MHz. En la Ilustración 2.6 se observa la canalización de las frecuencias descritas anteriormente.

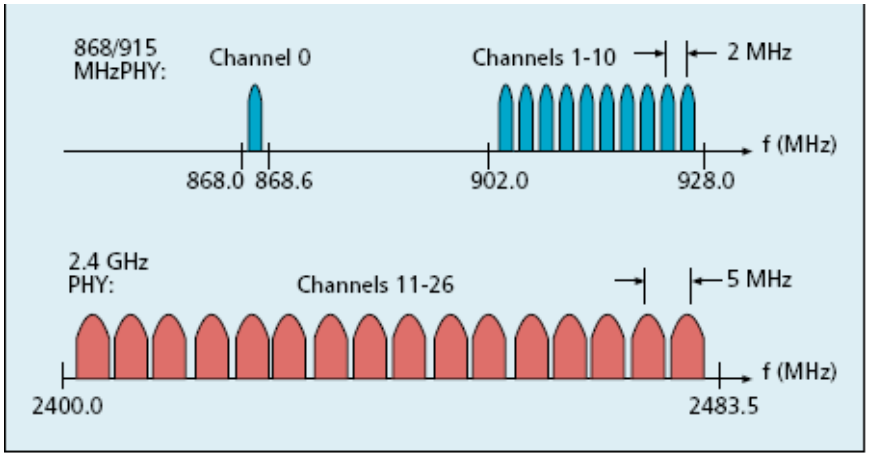


Ilustración 2.6: Estructura de canales

Subcapa MAC

Esta subcapa proporciona dos servicios: el servicio de datos y el servicio de gestión MAC. Debe definir los modelos de asociación y disociación, la entrega de la trama, los mecanismos de acceso al canal, validación de la trama, garantía del manejo de las ranuras de tiempo así como el manejo de los *beacons,* pequeños paquetes que son enviados a toda la red para informar del estado del nodo.

Las subcapas MAC proporcionan dos tipos de servicios hacia capas superiores que se acceden a través de dos puntos de acceso a servicios (SAP, *Servicie Access Point).* Los servicios de datos MAC se acceden por medio de la parte común de la subcapa (MCPS-SAP, *MAC Common Part Sublayer-Service Access Point*), y el manejo de servicios MAC se accede por medio la capa MAC de manejo de identidades (MLME-SAP, *MAC Sublayer Management Entity- Service Access Point*). Esos dos servicios proporcionan una interface entre las subcapas de convergencia de servicios específicos (SSCS, *Service-Specific Convergence Sublayer*) , la lógica de control de conexión (LLC, *Logical Link Control*) y las capas físicas. La relación entre estos servicios se observa en la Ilustración 2.7 . El conjunto de servicios MAC tiene 26 primitivas, haciéndolo muy versátil para las aplicaciones para las que está orientado.

De forma resumida, las principales características de esta capa son:

- Gestión de *beacons.*
- Acceso al canal.
- Gestión del GTS. (*Guaranteed time slots,* Ranuras de tiempo garantizado)[7]
- Validación de trama. (*frame*)
- Envío de trama de reconocimiento. (*ack*)

- Asociación y disasociación.
- Además proporciona mecanismos para la implementación de aplicaciones de seguridad.

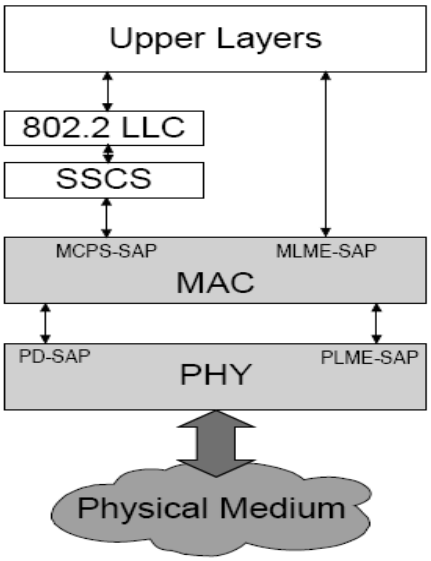


Ilustración 2.7: Arquitectura de la LR-WPAN

Estructura del *Superframe*

El estándar IEEE 802.15.4 permite el uso opcional de una estructura de *superframe,* esta estructura define la organización de tiempos y tramas para entablar una comunicación. El formato del *superframe* está definido por el coordinador. El *superframe* esta acotado por los *beacons* de la red enviados por el coordinador y está divida en 16 trozos de igual tamaño. En el *superframe* es posible encontrar una región activa y otra inactiva que los coordinadores usan para ahorrar energía pasando a modo suspendido. Un ejemplo del contenido del *superframe* se observa en la Ilustración 2.8. El *beacon* es transmitido siempre en la primera ranura de tiempo del *superframe*, y es usado para proporcionar sincronización, identificar la red, y describir la estructura de los *superframes*. Cualquier dispositivo que quiera comunicarse durante el periodo de CAP debe hacerlo usando el mecanismo CSMA-CA (*Carrier Sense Multiple Access with Collision Avoidance,* Acceso múltiple por detección de portadora con evasión de colisiones) [3].

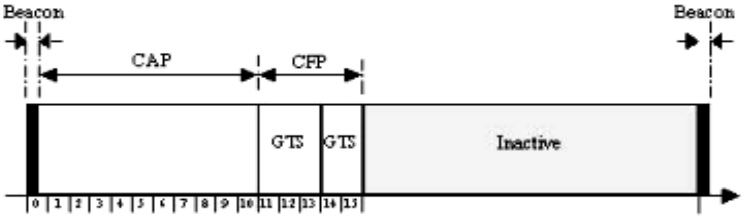


Ilustración 2.8 MAC super-frame

Para aplicaciones de baja latencia o aplicaciones que requieren un ancho de banda específico, el coordinador puede dedicar porciones de la región activa para esa aplicación, estas porciones son las llamadas GTS (*Guaranteed time slots*) [7].

2.3.4 Funcionamiento

La capa MAC define dos tipos de nodos, los FFD (*Full Function Devices*) y los RFS (*Reduced Function Devices*). Los FFD están equipados con todas las funciones de las capa MAC, lo que les permite actuar como coordinadores en una red o como dispositivos finales. Cuando actúan como dispositivos coordinadores se encargan de enviar *beacons* que proporcionan servicios de sincronización, comunicación, así como los mecanismos de unión a la red. Los dispositivos del tipo RFD que solo pueden actuar como dispositivos finales suelen ir equipados con sensores y/o actuadores pudiendo interactuar solo con los FFD.

En el estándar se definen dos tipos de redes: topología en estrella y comunicación *peer-to-peer* (comunicación punto a punto). En la topología en estrella se adopta un modelo de maestro – esclavo, donde un dispositivo FFD toma el rol de maestro pasando a ser el coordinador de la red (Coordinador de PAN), y los demás dispositivos, ya sean FFD o RFD serán dispositivos finales los cuales solo podrán comunicarse con el coordinador de la red. En la topología de red *peer-to-peer* los dispositivos FFD pueden comunicarse con los otros dispositivos FFD que tenga en el alcance de su radio o fuera de esta a través de otros FFD que actuarán como intermediarios, formando lo que se conoce como redes multisalto (*Multihop Network*).

El coordinador de la red puede operar su red (PAN) usando o no *superframe*s. En el caso de usar *superframes* este empieza enviando un *beacon* que sirve para propósitos de sincronización así como para describir la estructura del *superframe* y enviar la información de control a la red. El *superframe* está dividido en una región activa y otra no activa donde el coordinador puede ponerse en modo suspendido (*sleep mode*) para ahorrar energía. La región activa está divida en *slots* de tamaño fijo y contiene el CAP (*Contention Access Period* – Periodo de acceso por contención), donde los nodos compiten por acceder al canal usando el protocolo de acceso al medio CSMA-CA, también en esta región está contenido el CFP (*Contention Free Period* – Periodo de libre acceso), donde los nodos transmiten en periodos de tiempo garantizados (GTS – *Guaranteed Time Slots*) sin competir por el acceso al canal, estos periodos de tiempo GTS son administrados por el coordinador de red.

Cuando un dispositivo final necesita enviar datos a un coordinador fuera del periodo de tiempo garantizado (GTS), debe de esperar al *beacon* de sincronización que emite el coordinador y después competir por el acceso al medio en la región de CAP. Por otra parte, la comunicación desde el coordinador hacia los dispositivos finales se realiza de

manera indirecta. Es decir, el coordinador almacena el mensaje y en su *beacon* anuncia que tiene mensajes pendientes de enviar. Normalmente los periodos de inactividad de los dispositivos finales son más largos que los de los coordinadores. Por eso cuando los dispositivos finales pasan a estar activos esperan la recepción del *beacon* comprobando así si tienen mensajes pendientes de recibir, en cuyo caso hacen una petición explícita al coordinador durante la región de CAP para que le envíe el mensaje. Cuando un coordinador quiere establecer comunicación con otro coordinador debe esperar hasta el *beacon* para sincronizarse con él y actuar como si fuera un dispositivo final.

Otra opción en la redes es no utilizar *superframes*. En este caso el coordinador de la red nunca envía *beacons* y la comunicación se realiza mediante CSMA-CA. El coordinador debe estar siempre activo para recibir datos desde algún dispositivo final. La comunicación desde el coordinador hasta los dispositivos finales, se produce por medio de sondeo por parte de los dispositivos finales, una vez que salen de sus periodos de latencia preguntan al coordinador si tiene envíos pendientes para ellos. En caso de que haya algún envío pendiente, el coordinador envía el mensaje o si no los hay envía una trama de respuesta indicando la ausencia de mensajes pendientes de envío. La comunicación entre coordinadores no representa problemas al estar ambos activos en todo momento.

Además de la transferencia de datos, la capa MAC ofrece servicios de escaneo de canales, así como funciones de asociación y disociación. El proceso de descubrir diversos canales lógicos se realiza enviando un *beacon* de solicitud y esperando la respuesta o simplemente escuchando los diferentes *beacons* de los nodos que están en su radio de acción para intentar localizar redes cercanas así como sus coordinadores. Las capas superiores son las que deciden a que PAN unirse, normalmente basadas en criterios de unirse a la red más cercana y con la que necesita emplear menos energía para la comunicación, o del nivel de saturación de los canales, buscando los canales que están menos utilizados y tras esto envían la petición de asociación a la capa MAC para que esta la lleve a cabo. Este proceso involucra el envío de solicitud de asociación hacia el coordinador de la red a la que se va a unir y esperar hasta que se reciba la aceptación, en caso de ser afirmativa también recibirá una dirección corta de 16bits para el direccionamiento dentro de la red, evitando así tener que usar la dirección de 64bits.

Formato de las tramas MAC

La estructura del *Frame* ha sido diseñada para mantener la mínima complejidad pero haciéndola suficientemente robusta para soportar las transmisiones en un canal ruidoso. Cada nueva capa añade a la estructura cabeceras y terminaciones específicas de esas capas. La estructura estándar define las siguientes 4 estructuras de trama.

- *Becaon Frame*, usado por el coordinador para trasmitir *beacons*.

- Trama de datos, usado para la transmisión de datos.
- Trama de confirmación (*acknowledgment*), usada para la confirmación satisfactoria de las tramas.
- Trama de comando MAC, usada para manejar todas las transferencias de control entre entidades MAC.

Beacon Frame

Estas tramas son transmitidas por el coordinador de red, en redes con *beacons* habilitados. En la parte de datos de la trama MAC (MAC *payload*), está contenida la especificación del *superframe*, los campos de GTS, los campos de direcciones pendientes, y la información del *beacon*. El tamaño del campo del MAC *payload* esta prefijado, en la cabecera de la MAC (MHR, *MAC Header*) , donde encontramos los campos del número de control de la trama, el número de secuencia y los campos de direcciones así como los bits de seguridad adicionales de la cabecera. Para terminar la trama tras el MAC *payload* encontramos dos bytes de finalización (MFR, *MAC Footer*) con lo que se comprueba la integridad de la trama.

La estructura de la trama de la capa física conteniendo un *beacon frame* incorporado en ella se puede encontrar en la Ilustración 2.9, donde se puede apreciar los diferentes campos que aporta la capa física como los que aporta el *Beacon frame*. Para ser enviada esta trama es incorporada en el campo de *Paylaod* de la capa física. Donde se le añade la cabecera de la capa física que contiene una cabecera de sincronización (SHR, *Synchronization Header*) dependiendo de la implementación del medio físico de transferencia, también y detrás de esta se añade la cabecera que indica el tamaño del *payload* contado en número de bytes (PHR, *Phy Header*). Esta cabecera es común a los medios físicos de transferencia.

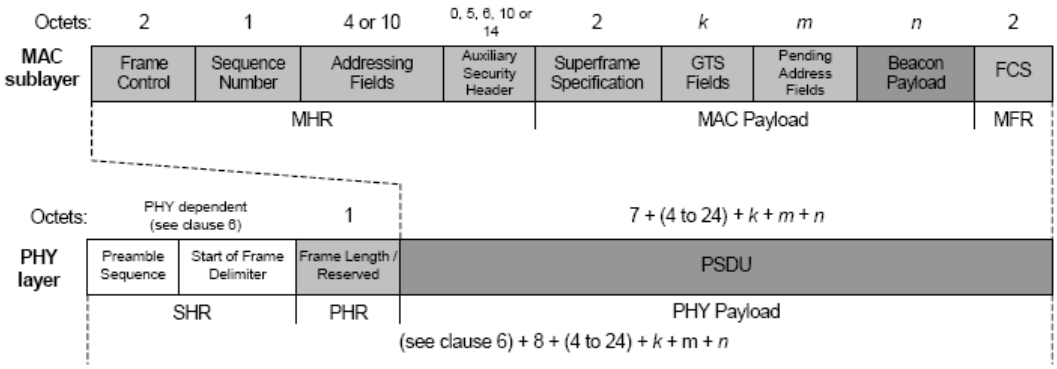


Ilustración 2.9 *Beacon frame* y su integración en la capa física

Trama de datos

Como se puede observar en la Ilustración 2.10, la trama de datos tiene un formato muy similar a la trama de *Beacon*, compuesta de una cabecera MHR, que contiene los mismos campos que esta, excepto por el campo de direcciones que ahora puede tener desde 4 hasta 20 bytes de longitud, dos bytes de finalización de trama que al igual que anteriormente contienen la información de chequeo de integridad, y la parte de la carga o *payload* de la subcapa *MAC*, donde aquí únicamente contiene un campo que es la información o datos que se quieren transmitir.

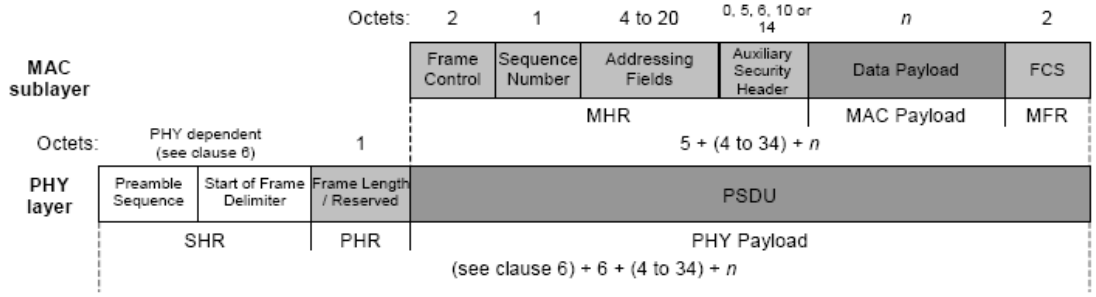


Ilustración 2.10: Trama de datos y su integración en la capa física

Al igual que en el resto de tramas se observa que esta trama queda insertada en el campo de *payload* de la trama de la capa física, donde en sus cabeceras especificamos el tamaño del *frame* resultante.

Trama de confirmación (*acknowledgment frame*)

La trama de confirmación mostrada en la Ilustración 2.11, tiene un tamaño fijo de 5 bytes, y está compuesta por una cabecera y un campo de finalización consistente en dos bytes de comprobación de integridad de la trama.

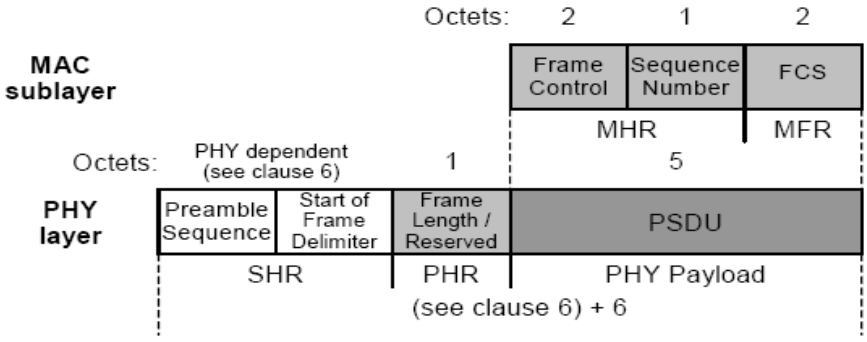


Ilustración 2.11: Trama de confirmación (Acknowledgment *frame*)

La cabecera está compuesta de dos campos uno de dos octetos conteniendo la identificación de la trama y otro de un octeto con el número de secuencia. Esta trama no tiene campo de *payload* en la subcapa MAC. Al igual que las anteriores tramas se integra en el campo de *payload* de la trama de la capa física para ser transmitido.

Trama de comando MAC

Esta trama está compuesta por tres campos, el primer campo de cabecera (MHR), que es igual que en la trama de datos, el campo de finalización compuesto por los 2 bytes de comprobación de integridad y el campo de *payload* de la subcapa MAC. Este campo a su vez esta divido en dos sub-campos, el campo de comando que tiene una longitud de un octeto donde se indicada el tipo de comando enviado o recibido, y el campo de comando donde va alojado el comando a transmitir, teniendo este último una longitud variable. Para ser enviado se incorpora toda la trama en el campo del *payload* de la trama de la capa física, quedando la trama resultante la mostrada en la Ilustración 2.12, donde en el campo PHR de la cabecera ha de especificarse el tamaño total de toda la trama física.

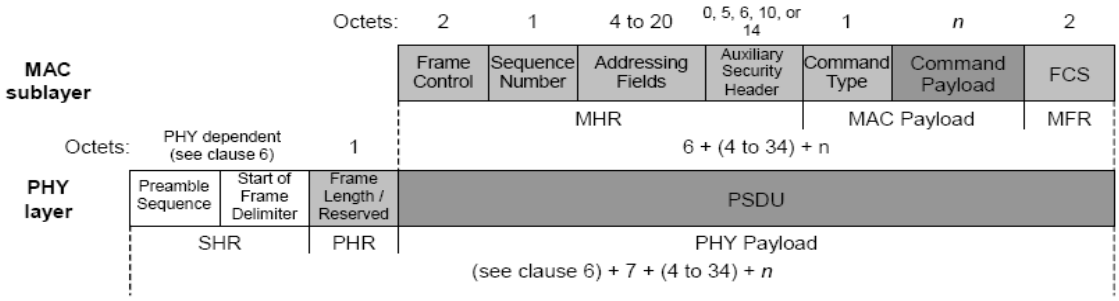


Ilustración 2.12: Formato de trama MAC

Tipos de transferencia de datos

En el estándar podemos encontrar tres esquemas de transferencia de datos, los cuales cumplen las siguientes necesidades: comunicación dispositivo final con coordinador, comunicación desde coordinador hacia dispositivo final y comunicación entre dos dispositivos en las redes *peer-to-peer*. Los dos primeros esquemas se usan en las redes en estrella, mientras que el último se usa en las redes peer-to-peer donde los nodos se comunican entre ellos punto a punto.

Los mecanismos de transferencia dependen de si la red en la que están soporta *beacons* o no. Una red con soporte de *beacon* se usa cuando esta requiere sincronización o soporte para dispositivos de baja latencia. Si la red no necesita de estas características el coordinador puede decidir no usar *beacons* para las transferencias, sin embargo los *beacons* son todavía necesarios para descubrir nuevas redes.

Transferencia desde dispositivo final a coordinador

Cuando un dispositivo quiere transmitir datos a un coordinador en un una red con *beacon*, el dispositivo se pone en modo escucha hasta recibir un *beacon* contenido en el *superframe* que envía el coordinador. En este momento el dispositivo final transmite los

datos usando una porción de tiempo mediante CSMA-CA. Tras realizar la transmisión el coordinador envía un reconocimiento del resultado de la transferencia. En definitiva se sigue el diagrama de comunicación de la Ilustración 2.13 -a.

En las redes sin *beacons* este esquema de comunicación, que se representa gráficamente en la Ilustración 2.13-b, es más sencillo. Al estar siempre el coordinador activo, solo se ha de transmitir los datos mediante el método de CSMA-CA no ranurado hacia el coordinador. El coordinador puede opcionalmente enviar un reconocimiento indicando que se ha completado la transferencia.

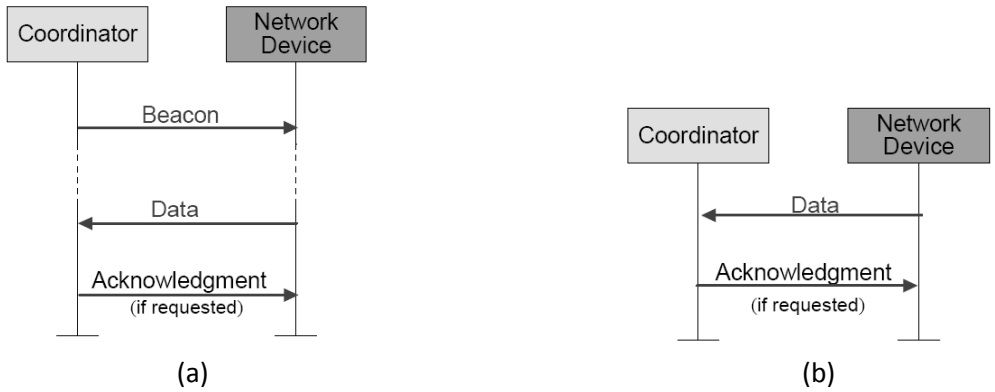


Ilustración 2.13: Esquema transferencia dispositivo final hacia coordinador, (a) -En una red con *beacons*, (b) - En una red sin *beacons*

Transferencia desde un coordinador a un dispositivo final

Cuando el coordinador quiere transferir datos hacia un dispositivo final en una red con *beacons*, el coordinador en vez de enviar el mensaje inmediatamente lo almacena. La siguiente vez que envía el *beacon* de inicio del *superframe* a su nodos asociados, en este les envía la notificación de que hay mensajes pendientes de enviar. Los dispositivos finales que periódicamente despiertan de sus estados de inactividad y escuchan al coordinador para ver si hay nuevos mensajes, al ver este hecho realizan al coordinador una petición de envío del mensaje usando CSMA-CA ranurado. El coordinador envía la confirmación de la recepción de la solicitud de datos. Tras esto, el coordinador envía los datos usando también el CSMA-CA y borra el mensaje que tenia almacenado y que ya ha enviado. Opcionalmente el dispositivo final puede enviar confirmación de la correcta recepción de los datos, la Ilustración 2.14 -a resume el flujo de comunicación.

En el caso de que red sea sin *beacons,* cuando el coordinador desea enviar información a un dispositivo final, éste guarda la información y queda a la espera de que un dispositivo final le haga una solicitud de envió de datos. Esto se produce cuando los dispositivos finales despiertan de sus periodos de inactividad y envían al coordinador un comando MAC de solicitud de datos usando CSMA-CA no ranurado. El coordinador notifica la correcta recepción de la petición. Si no tiene datos pendientes de envío para ese

dispositivo se lo notifica al dispositivo final mediante la misma notificación o con un nuevo *frame* con el campo de datos con tamaño cero. En caso de haber mensajes pendientes, el coordinador genera una nueva trama con los datos y los envía al dispositivo final mediante CSMA-CA no ranurado. Una vez que el dispositivo final recibe correctamente los datos envía un reconocimiento de operación terminada con éxito. En la Ilustración 2.14-b se observa el flujo de las comunicaciones descritas.

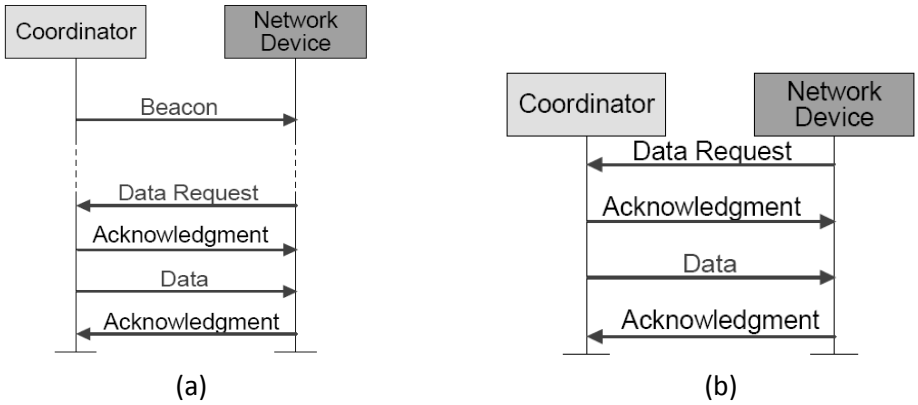


Ilustración 2.14: Esquema transferencia coordinador hacia dispositivo final, (a) - En una red con *beacons,* (b) - En una red sin *beacons*

Transferencia peer-to-peer

En una red peer-to-peer, cada dispositivo puede comunicarse con cualquier otro dispositivo que se encuentre localizado dentro de la cobertura de su radio. Para que este procedimiento sea efectivo los dispositivos que van a comunicarse deben de estar sincronizados constantemente unos con otros. Las transmisiones de los datos se producen usando CSMA-CA no ranurado, los procedimientos de sincronización requiere de técnicas más complejas.

2.3.5 Otros Protocolos Basados en el IEEE 802.15.4

Basados en el estándar IEEE 802.15.4 y de nivel superior a este se encuentran entre los más usados los protocolos 6lowPAN y ZigBee, a continuación se definen y se describen las principales características de cada uno de ellos, así como, sus principales aplicaciones para las que han sido creados.

ZigBee

ZigBee es el nombre de la especificación de un conjunto de protocolos de alto nivel de comunicación inalámbrica para su utilización con radios de bajo consumo, basada en el estándar IEEE 802.15.4 de redes inalámbricas de área personal (WPAN). Su objetivo son las aplicaciones que requieren comunicaciones seguras con baja tasa de envío de datos y maximización de la vida útil de sus baterías.

Una de sus principales aplicaciones es el área de la domótica, debido a su bajo consumo, su sistema de comunicaciones vía radio, con topología de red en malla, y su fácil integración. El nodo ZigBee más completo requiere en teoría cerca del 10% del software de un nodo Bluetooth o WiFi típico. Esta cifra baja al 2% para los nodos más sencillos.

Las funciones de la pila de protocolo ZigBee, la cual se representa en la Ilustración 2.15, son: proporcionar los protocolos de enrutamiento, el mantenimiento de la topología, detección de los vecinos, preservación de la energía de la red, estableciendo periodos de bajo consumo (*sleep)* en los nodos además de proporcionar tolerancia a fallos y escalabilidad.

Las principales características que ZigBee presenta son:

- Bajo consumo.
- Topología en malla, por lo que si un nodo cae, los paquetes son direccionados a través de otros nodos de la red para llegar al host.
- Fácil integración, lo que abarata costes de producción.

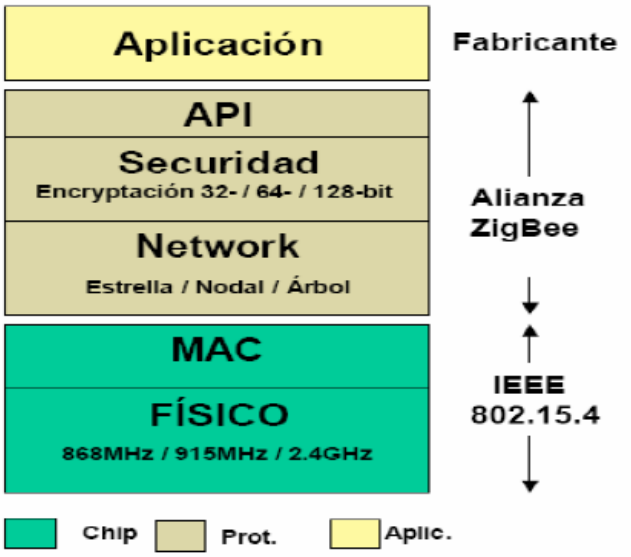


Ilustración 2.15: Capas del protocolo ZigBee

6lowPAN

6lowpan [8] es un acrónimo de IPv6 [9] sobre redes inalámbricas de área personal de bajo consumo. 6lowpan es el nombre del grupo de trabajo de este protocolo que define la encapsulación y los mecanismos de compresión de las cabeceras que permiten que los paquetes IPv6 sean enviados y recibidos desde redes basadas en IEEE 802.15.4.

El objetivo de tener el protocolo IP en redes inalámbricas de bajo consumo es para las aplicaciones que necesitan de conectividad a Internet con baja transferencia de datos en dispositivos de muy bajo consumo.

Como inconvenientes de crear paquetes de IPv6 sobre estas redes serian:

- La gran diferencia del tamaño de los paquetes entre ambas redes.
- Problemas con la dirección, ya que el IEEE 802.15.4 está orientado a trabajar en PANs con dirección de 64 o 16 bits e IPv6 necesita IPs de 128 bits en todo caso.
- El protocolo IP no está orientado para el bajo consumo. Está optimizado para resolver problemas de tráfico y congestión pero no lo está para tener baja transferencia de datos y correr en dispositivos de características limitadas.

2.3.6 Sistemas Operativos para Redes Inalámbricas de Sensores

Los sistemas operativos que corren apoyados en el IEEE 802.15.4 son menos complejos que los encontrados para PC, en gran parte debido a sus diferencias de uso. Estos sistemas para WSN no necesitan una gran interacción con los usuarios. Además, en los motes el sistema ha de consumir pocos recursos y ser eficiente en la gestión de la energía.

Como ventajas encontramos el que se proporcionan métodos para acceder al hardware independientemente de la plataforma, permitiendo desarrollar aplicaciones que funcionen en diferentes motes. Los sistemas operativos más conocidos y utilizados serían, TinyOs[10], Contiki[11], LiteOs[12] aunque se pude encontrar otros de nuevo desarrollo como BTnut[13], Nano-RK[14] y Mantis[15].

TinyOs

TinyOS es un sistema operativo de código abierto basado en componentes, para redes de sensores inalámbricas. El sistema y sus bibliotecas están escritas en el lenguaje de programación nesC [16] como un conjunto de tareas y procesos que colaboran entre sí. Está diseñado para incorporar nuevas innovaciones rápidamente y para funcionar bajo las importantes restricciones de memoria que se dan en las redes de sensores. TinyOS está desarrollado por un consorcio liderado por la Universidad de California en Berkley en cooperación con Intel Research y Crossbow Technology.

TinyOS tiene un núcleo reducido multitarea, del tipo *Event-driven*, es decir, que funciona a partir de eventos que llaman a funciones. Actualmente soporta la programación sobre diferentes procesadores y permite programar cada tipo con un único identificador para diferenciarlo. Esto hace posible que se pueda compilar en diferentes plataformas con solo cambiar el atributo de este identificador.

En su programación los componentes se enlazan a través de sus interfaces, las cuales son bidireccionales y especifican un conjunto de funciones que están implementadas bien por los proveedores o bien por los usuarios. TinyOs ya provee interfaces y componentes para los elementos más comunes como paquetes de comunicación, ruteo, sensorización, actuación y almacenamiento. El lenguaje NesC tiene como particularidad que genera un ejecutable que contiene todo lo que necesita para ejecutarse como bibliotecas, manejadores de las interrupciones y demás elementos con los que enlaza.

Contiki

Contiki es un pequeño sistema operativo de ordenador de código abierto desarrollado para uso en un número de pequeños sistemas pasando desde ordenadores de 8-bit a sistemas integrados sobre microcontroladores, incluyendo nodos de redes de sensores. El nombre Contiki viene de la famosa balsa Kon-Tiki, realizada con troncos de madera balsa siguiendo las técnicas de construcción indígenas y que fue utilizada por el explorador Thor Heyerdahl para demostrar que pudo ser posible el viaje desde Sudamérica hasta la Polinesia en tiempos Precolombinos.

A pesar de la multitarea que provee y la pila TCP/IP incluida, Contiki sólo requiere unos kilobytes de código y unos cientos de bytes de RAM. Un sistema totalmente completo con una GUI requiere aproximadamente 30 kilobytes de memoria. El tamaño del código está en el orden de los kilobytes y el uso de la memoria puede configurarse para que sea de sólo unas decenas de bytes. Una configuración típica de Contiki consta de 2 KB de RAM y 40 KB de ROM. Dado que está diseñado para sistemas embebidos con escasa memoria funciona en una gran variedad de plataformas, desde microcontroladores embebidos, como el MSP430 y el AVR, a viejas computadoras domésticas.

Contiki consiste en un núcleo orientado a eventos, el cual hace uso de protohilos que son hilos sin pila propia lo que rebaja la memoria necesaria para ejecutar varios procesos a la vez, sobre el cual los programas son cargados y descargados dinámicamente. También soporta multihilado apropiativo por proceso, es decir, cuando un hilo obtiene la ejecución la mantiene hasta que se completan las tareas del hilo, comunicación entre procesos mediante paso de mensajes a través de eventos, al igual que un subsistema GUI (*Graphical User Interface*) opcional, el cual puede usar un soporte directo de gráficos para terminales locales, terminales virtuales en red mediante VNC [17] o sobre Telnet [18].

2.4 Soluciones Hardware para Redes Inalámbricas de Sensores

2.4.1 Introducción

En este apartado se explica el componente fundamental de toda red de sensores: sus nodos llamados Motes, los cuales disponen de un simple sensor o varios de ellos o incluso en algunos modelos incluyendo receptores de GPS. En el siguiente apartado se comienza con la descripción de la unidad principal: el mote, seguido de las diversas soluciones comerciales que los fabricantes basándose en este estándar han ido sacando al mercado. Posteriormente se describirán los principales kits de desarrollo existentes para introducirse en las WSN basadas en el estándar IEEE 802.15.4.

2.4.2 Motes

Los motes son los nodos que componen una red de sensores. Normalmente están equipados con un transmisor de radio u otro dispositivo de comunicaciones inalámbricas, un pequeño microcontrolador, y una fuente de energía, por lo general una batería. Mote es el nombre dado por los científicos de la universidad de Berkeley, en la participación en un proyecto, conocido como *Smart Dust* [19]. El objetivo del proyecto *Smart Dust* era reducir el tamaño del dispositivo hasta el tamaño de una mota polvo. Se exponen a continuación las características de los motes comerciales más ampliamente utilizados.

TelosB

Los motes TelosB son una plataforma para aplicaciones en redes de sensores de muy bajo consumo y alta recolección de datos. Llevan integrados tanto los sensores como la radio, antena, memoria y microcontrolador y pueden ser fácilmente programados. En la Ilustración 2.16 se muestra un diagrama de bloques donde es posible encontrar las interconexiones entre estos elementos.

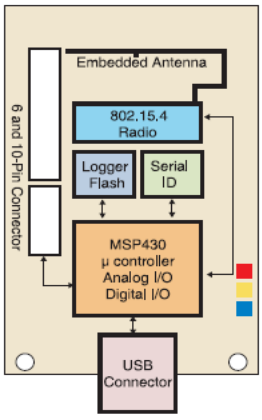


Ilustración 2.16: Diagrama de bloques del mote TelosB

Las operaciones de baja energía en los TelosB se llevan a cabo gracias al microcontrolador MSP430 F1611. Este procesador RISC de 16 bits consume muy poca energía tanto en el estado activo como durante el modo *sleep*. Para reducir al máximo este consumo, permanece en modo *sleep* durante la mayoría del tiempo. Se activa tan rápido

como puede para procesar y enviar, una vez terminado entonces vuelve a *sleep*. Utiliza un controlador USB del fabricante FTDI [20] para comunicarse con el ordenador y lleva el módulo de radio CC2420, el cual envía y recibe bajo el estándar IEEE 802.15.4. La radio tiene la posibilidad de enviar datos a muy alta frecuencia. El microcontrolador se comunica con la antena a través del puerto SPI y puede apagarlo para el modo de baja energía. La antena interna "Invertid –F micro strip" es una antena pseudo-omnidireccional que puede alcanzar los 50 metros dentro de un edificio y los 125 en el exterior. En la Ilustración 2.17 se puede observar el reducido tamaño de este mote, donde el mayor espacio se usa para el conector USB necesario para su programación y para el alojamiento de las baterías que irían en la parte posterior.

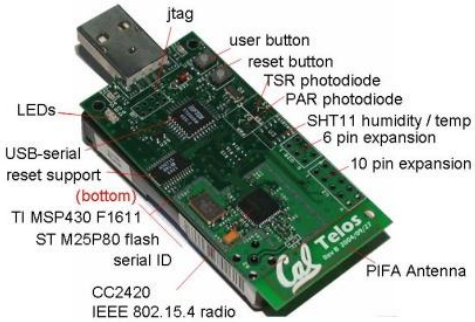


Ilustración 2.17: Mote TelosB

Las características esenciales del mote TelosB son:

- Transmisor Chipcon inalámbrico de 250Kbps 2.4GHz IEEE 802.15.4.
- Puede interactuar con otros dispositivos IEEE 802.15.4.
- Microcontrolador MSP430 de 8MHz. (10Kb deRAM y 48 Kb de Flash)
- ADC, DAC, supervisor de voltaje y controladora DMA integrada.
- Antena, sensores de humedad (0-100% RH), temperatura.(-40 ºC – 123,8 ºC) y radiación (320 nm - 730 nm)
- Muy bajo consumo. (1,8 mA en estado activo y 5,1 µA en estado *sleep*)
- Rápido en despertar del modo *sleep*. (<6 µs)
- Hardware para encriptación y autenticación de la capa de enlace.
- Programación y recogida de datos por USB.
- 16 pines para soportar una tarjeta de expansión y conector de antena opcional.

- Conector de antena SMA.
- Ayuda de TinyOS: enrutamiento de malla e implementación de las comunicaciones.

Los sensores de humedad/temperatura están fabricados por Sensiron AG[21], producidos con procesadores CMOS y emparejados con un dispositivo ADC de 14 bits. Los coeficientes para estos sensores se guardan en la EEPROM. La precisión del sensor de temperatura es de ±0.5 º C y el de humedad de ±3.5% RH.

Esta plataforma consigue un bajo consumo de potencia permitiendo una larga vida a las baterías además de tener un tiempo mínimo en el estado de *wakeup*, otro de los objetivos dentro de las estrategias de bajo consumo.

Los TelosB se alimentan con 2 baterías del tipo AA, aunque si está conectado mediante el puerto USB para programación o comunicación, la alimentación la proporciona el ordenador. También proporcionan la capacidad de añadir dispositivos adicionales mediante los dos conectores de expansión de los que disponen, estos pueden ser configurados para controlar sensores analógicos, periféricos digitales y displays LCD.

Motes MICAz

Usando la banda de frecuencias libres de 2.4GHz y cumpliendo la especificación del IEEE 802.15.4, Crossbow Technology empresa fundada en 1995 y especializada en redes de sensores, ofrece un mote que puede ser usado para formar redes inalámbricas de sensores de bajo consumo. En la Ilustración 2.18 se puede ver el aspecto exterior de este Mote, donde al igual que en modelos de otros fabricantes el mayor espacio es el ocupado por las 2 baterías del tipo AA.

Ilustración 2.18: Mote Micaz

Las principales características del mote Micaz son:

- Transceptor cumpliendo la especificación IEEE 802.15.4.
- Funciona en la banda ISM (*Industrial, Scientific and Medical*) desde 2.4GHz hasta 2.4835 GHz.

- Utiliza la técnica de modulación conocida como DSSS (*Direct Sequence Spread Spectrum*) la cuál es resistente a las interferencias de radiofrecuencia y provee de esta manera seguridad en los datos de sus transmisiones.
- Puede llegar alcanzar tasas de datos de hasta 250 Kbps.
- Preinstalación de TinyOS.
- "Plug and Play" de las diferentes placas de sensores ofrecidas por crossbow. Entre estas destacan placas de entradas analógicas, digitales, soporte de I2C, SPI y UART. (*Universal Asynchronous Receiver-Transmitter*)

Entre sus placas de expansión se encuentra la MIB510CA, mostrada en la Ilustración 2.19, esta placa proporciona un conector JTAG para permitir la depuración del código y un puerto serie con conector Sub-D DB9,para poder conectarla al PC. Este puerto serie sirve para la programación de los dispositivos, así como para conectar al PC un nodo sumidero.

Ilustración 2.19: Placa de expansión MIB510CA para MICAz

IMOTE

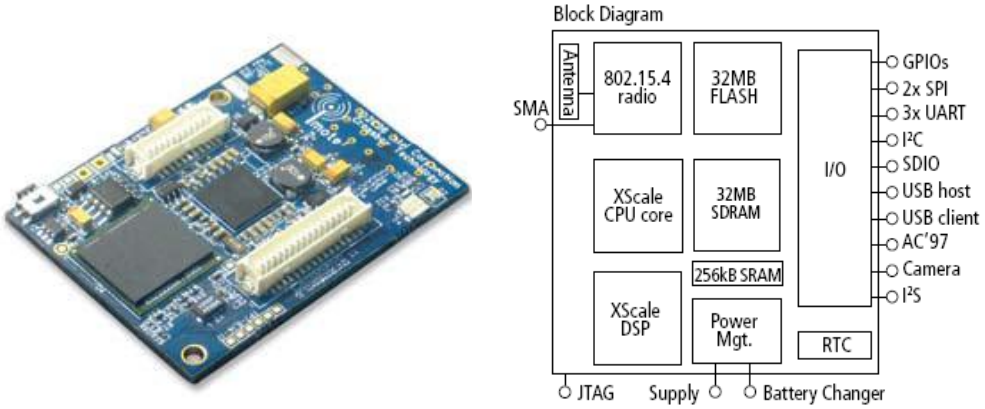
Originalmente diseñado por Intel, fue adquirido por la compañía Crossbow en 2007 y es la empresa que actualmente lo distribuye. Este mote se diferencia de la mayoría de los existentes en el mercado por usar un procesador XScale, funcionando a 416MHz, con una potencia superior a los actuales Atmel ATmega128, que tienen una velocidad de 16MHz. La memoria de este mote también es superior al resto de motes en el mercado siendo de decenas de megas, mientras que otros motes tienen montados unos cientos KB.

Es por esto que este mote es muy indicado para sensores que conlleven más complejidad computacional, como por ejemplo, sensores inerciales, de vibración y detección de movimientos sísmicos. Este mote consta de estas características por haber sido orientado al ámbito industrial más que al medioambiental donde los requerimientos de los sensores suelen ser menores.

Las principales características del IMOTE son:

- Procesador Marvell PXA271 XScale de 13 – 416 MHz.
- Co-procesador Marvell, Radio MMX DSP.
- 256K de SRAM, 32 MB de FLASH y 32 MB de SDRAM.
- Dispone de indicador de estado LED multicolor.
- Antena Integrada de 2.4 GHz, con capacidad de alcance de hasta 30 metros, para conseguir una distancia mayor se puede utilizar un conector de SMA.
- Permite comunicación con dispositivos USB.
- Gran cantidad de E/S estándar, 3XUART, 2XSPI, I2C, SDIO, GPIOs.
- Algunas de sus aplicaciones son: monitorización y análisis industrial, monitorización sísmica y de vibraciones.
- Es una plataforma avanzada con diseño modular.
- El procesador tiene dos modos de funcionamiento en bajo consumo: "sleep" y "deep sleep."
- El procesador cuenta con numerosos temporizadores, entre ellos un reloj en tiempo real.
- Usa el transceptor de radio CC2420 IEEE 802.15.4.
- Velocidad de transmisión de datos a 250 Kb/s con 16 canales a 2.4 GHz.

En la Ilustración 2.20 se muestra su placa y un diagrama de bloques de sus componentes.



(a) (b)

Ilustración 2.20: a) Vista exterior del IMote, b) Diagrama de bloques del IMOTE

Waspmote

La empresa Libelium[22] (Spin-off de la Universidad de Zaragoza) lanzó en el 2009 su plataforma Waspmote, en el cuál centraron sus esfuerzos en conseguir una plataforma con muy bajo consumo, llegando a los 0.7 uA en modo hibernación. Esta plataforma usa los módulos de radio XBee. En la Ilustración 2.21 se muestra el mote con su módulo de radio, entrada USB, tarjeta SIM, lector de tarjetas MMC/SD y el conector de expansión.

Ilustración 2.21: Waspmote en placa de desarrollo

Esta plataforma tiene dos modelos diferentes de radio: el XBee y el XBee Pro que le permite trabajar en las siguientes frecuencias y protocolos:

- Frecuencias: 2.4Ghz, 900 MHz, 868MHz.
- Protocolo: IEEE 802.15.4, ZigBee.
- Potencia: Desde 1mW a 100mW.

Estas radios permiten obtener rangos de operación de hasta 7 kilómetros en la banda de 2.4GHz, 24 km en la de 900MHz y 40 km en la banda de 868MHz. Estos módulos llevan incorporados un acelerómetro de 3 ejes en su placa. Además están basados en una arquitectura modular con lo cual es sencillo ampliar sus capacidades. Siendo algunos de los módulos ampliables más utilizados los de GPS, GPRS, lectores de tarjetas SD, Sensores para gases, diferentes sensores de luminosidad, vibración, nivel de líquidos y módulos de entrada salida genéricos como ADC, entradas digitales, etc.

2.4.3 Kits de Desarrollo

Por ser los kits de desarrollo un elemento importante en las primeras fases de desarrollo de un proyecto así como para evaluar la idoneidad de un mote para la aplicación a desarrollar, a continuación se describirán brevemente los diferentes kits de desarrollo basados en el estándar IEEE 802.15.4 de los principales fabricantes.

ZigBit development kit

Meshnetics[23], adquirida por Atmel ofrece, entre otros, el kit de desarrollo "ZigBit".

Ilustración 2.22: Kit de desarrollo de Meshnetics

El kit el cual podemos ver en la Ilustración 2.22, está formado por:

- 3 placas de desarrollo, con diferente toma de antena.
- 3 cables USB (USB para la conexión de los módulos con el PC de configuración).
- 1 CD-ROM con software y documentación.

Las principales características de este kit de desarrollo son:

- Permite la programación embebida, con lo que los fabricantes de equipos pueden introducir sus propios desarrollos en el módulo a través de su ANSI C API.
- Un completo juego de aplicaciones para testear el funcionamiento (plantilla de aplicaciones), como programas para la configuración serie, monitorización de las medidas de los sensores incorporados, etc.
- Soporte para la incorporación de sensores de otros fabricantes.
- Completo juego de APIs y drivers para periféricos serie, I2C, ADC y 1-cable.
- Todo ello bajo el intuitivo entorno de programación de Atmel, que permite incluso depuración in-situ.
- Permite la programación mediante comandos AT en su puerto serie.

Este kit se suministra además con el software WSN Monitor (*Wireless Sensor Network Monitor*), mostrado en la Ilustración 2.23, que permite de una forma rápida e intuitiva ver la red, así como ver la información aportada por cada sensor y la indicación de la calidad de su conexión (LQI).

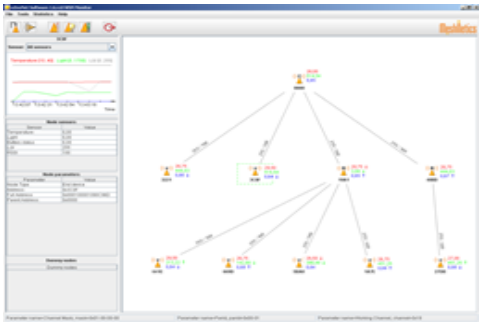


Ilustración 2.23: Software WSN Monitor

Microchip

La casa microchip ofrece, entre otros, el PICDEM Z como kit de desarrollo inalámbrico, bajo el nombre de "*PICDEM Z Demonstration Kit*" [24] con número de referencia DM163027.

El kit, mostrado en la Ilustración 2.24, está formado por los siguientes elementos:

- Dos placas de desarrollo con dos transceptores RF de radio MRF24J40MA
- Dos conectores de 6 pines para conectar la placa directamente con el software Microchip MPLAB ICD 2.
- Código fuente de la pila de protocolo ZigBee.
- CD-ROM con las guías de usuario.

Ilustración 2.24: Kit de desarrollo PICDEM Z

Este kit de desarrollo se presenta con las siguientes características:

- Usa el Microchip's MRF24J40 transceptor, como módulo de radio, usando la banda de 2.4 GHz, implementando el estándar IEEE 802.15.4.
- Dispone de un microcontrolador de la familia PIC 18. Concretamente esta placa utiliza el PIC18LF4620 MCU con 64 KB de memoria Flash.
- Presenta una pila de protocolo ZigBit preinstalada, soportando RFD (Reduced *function device*), FFD (*Full Function Device*) y Coordinador.
- Conector modular de 6 pines para interactuar directamente con MPLAB mediante ICD 2, y permitir la programación desde este software.
- El software ZENA™ para análisis de la red inalámbrica creada.
- Las placas de pruebas del kit están equipada con sensor de temperatura (Microchip TC77), LEDs e interruptores.
- Incorporan interface de comunicación RS-232 y regulador de tensión de 9V a 3.3V.

Texas Instruments

Texas Instruments lanzó al mercado a principios de 2008 el kit denominado "eZ430-RF2480"[25].

Este kit, mostrado en la Ilustración 2.25, dispone de:

- 3 Placas de comunicaciones con módulo de radio CC2480.
- 2 Placas AAA eZ430 con alimentación por batería. (Sin USB). Como la mostrada en la Ilustración 2.25–a.
- 1 Placa de emulador con USB, eZ430 (Para la programación y servidor host). Se muestra en la Ilustración 2.25-b.

(a) (b)

Ilustración 2.25: Placas del kit de desarrollo EZ430-RF2480, (a) - Placa AAA eZ430, (b) -Placa eZ430

El kit eZ430-RF2480 es un sistema microcontrolador con soporte USB basado en el MSP430 y usando el popular módulo de radio en la frecuencia de 2.4Ghz de TI, CC2480. Tiene todo lo necesario para evaluar este módulo de radio en poco tiempo y con el mínimo esfuerzo.

El kit utiliza una versión libre del entorno de desarrollo "IAR *Embedded Workbench Kickstart* "[26] el cuál es usado para la familias de microcontroladores MSP430, y permite escribir, compilar y depurar las aplicaciones.

Las principales características de este kit son las siguientes:

- ZASA ,*ZigBee Accelerator Sample Application,* aplicación de demostración para PC ya desarrollada, para la monitorización de la temperatura y el nivel de la batería a través de los sensores que lleve incorporados los motes.
- Código de ejemplo para demostrar la configuración, la selección de red y la comunicación entre motes.
- 5 pines GPIO (Pines de propósito general de entrada/salida).
- Alta integración y muy bajo consumo.
- Dos pines de entrada/salida digital generales conectados a dos LEDs rojo y verde para una visualización rápida de esas salidas.
- Pulsador generador de interrupción para aplicaciones de control.
- Área mínima al tener la antena integrada en el propio chip.

La placa de comunicaciones CC2480 se ha diseñado para cumplir las especificaciones Z-Accel[27] de TI, que imponen un sistema de comunicaciones de radio basado en el protocolo de puerto serie, aislando así la parte de radio como un sistema de caja negra con un API bien definida. La placa con USB eZ430, proporciona una conexión serie a través del USB con el PC, además de recibir la alimentación por medio de este. El resto de placas de eZ430 que se alimentan a baterías permite tener movilidad con ellos y situarlos donde la aplicación lo requiera.

Crossbow

La empresa Crossbow, líder en el mercado de las redes inalámbricas de sensores, tiene entre otros en el mercado el "WSN Imote2 Builder Kit", basado en el IMote2. La característica más resaltables de estos motes es su sistema operativo, que por defecto tiene instalado, está basado en la tecnología "Microsoft Framework .NET" y por tanto su metodología de programación se realiza con el software Microsoft Visual Studio.

El contenido del kit se puede ver en la Ilustración 2.26 , el cual está compuesto de:

- 3 Motes IMOTE2 pre-programados con la plataforma de desarrollo Microsoft .NET Micro Framework SDK.(IPR2410CA)
- 2 Placas de sensor. (ITS400CA)
- Copia de evaluación de Microsoft Visual Studio.

Ilustración 2.26IMOTE2 de Crossbow

Discovery kit

Zolertia ofrece el Dicovery Kit Z1 [28]. Este kit viene preparado para la instalación del sistema operativo TinyOs y Contiki, además de tener soporte para 6LoWPan. Una característica a destacar del Zolteria Z1, el mote del kit, es la incorporación de los *Phidget Port* [30], lo que permite la fácil conexión de varios sensores a la placa a través de estos conectores.

El kit Zolteria Discovery Kit Z1, incluye: 3 módulos Z1 con sus adaptadores para las baterías, 2 cables USB y el pack de baterías necesarias. El contenido se muestra en la Ilustración 2.27 .

Ilustración 2.27: Contenido de "Discovery kit"

El módulo Z1 está equipado con dos sensores digitales integrados en la misma placa (acelerómetro de 3 ejes y sensor de temperatura), es posible ver un detalle de este módulo en Ilustración 2.28 .

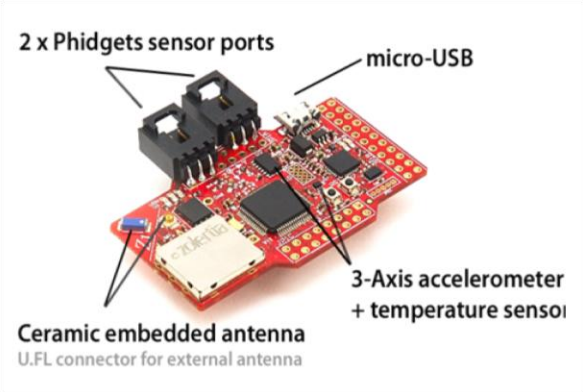


Ilustración 2.28: Placa de desarrollo "Discovery kit"

Las principales características de este kit de desarrollo son:

- Implementación del protocolo IEEE 802.15.4.
- Conector RP-SMA.
- Potenciómetro para entrada analógica.
- 2 Puertos. (5V y 3V regulados)
- Conector de expansión de 54 pines para GPIO, buses digitales, entradas y salidas analógicas e interrupciones.
- ADC, UART, I2C, SPI.
- Puerto Ziglet, está basado en I2C y permite la conexión de más de dos sensores a la placa de desarrollo.
- Conector USB.
- LED Tricolor.

XBee y XBeePro starter development kit

XBee/XBee Pro es el más conocido kit de desarrollo bajo el estándar IEEE 802.15.4 presentado por la casa Digi (anteriormente Maxstream) [31]. Este kit destaca por su configuración por defecto de fábrica que permite usarlo como un simple reemplazo de una conexión serie, además de ya traer el adaptador cruzado de DB9, para poder realizar las pruebas de alcance del la radio con solo conectarlo a la placa alimentada por batería.

El contenido del kit, el cual se puede ver en la Ilustración 2.29, está compuesto por dos módulos XBee y dos placas de interface con comunicación serie. Aunque también se pueden encontrar placas de interface con comunicación a través de USB.

Ilustración 2.29: Kit de desarrollo XBee/ XBee Pro

Contenido detallado del kit de desarrollo:

- Placa de desarrollo con conector RS232.
- Cable RS232.
- Placa de desarrollo con conector USB.
- Cable USB.
- Adaptador de corriente 9V 1A.
- Conector de batería 9V.
- Adaptador Serial *loopback*. Ver Ilustración 2.30.
- Guía de inicio.
- CD.

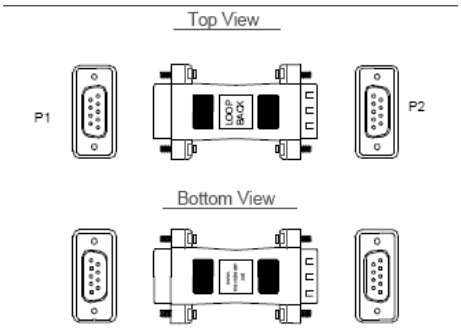


Ilustración 2.30: Adaptador serial loopback

Estas placas de interface basadas en el módulo XBee (ver Ilustración 2.31) contienen LEDs que permite ver la calidad de la señal de transmisión (comportamiento por defecto), monitorización de salidas y/o entradas, pulsadores para las entradas digitales, conectores de expansión que permiten tener acceso a cada uno de los pines del modulo, así como del adaptador de corriente, y los conversores de adaptación de nivel de las señales del puerto serie. En el caso de la placa con puerto RS232, el adaptador convierte señales TTL del módulo XBee a señales RS232 y en la placa USB convierte las señales TTL serie a un puerto Serie-USB emulado, se pueden ver ambas placas en la Ilustración 2.32 .

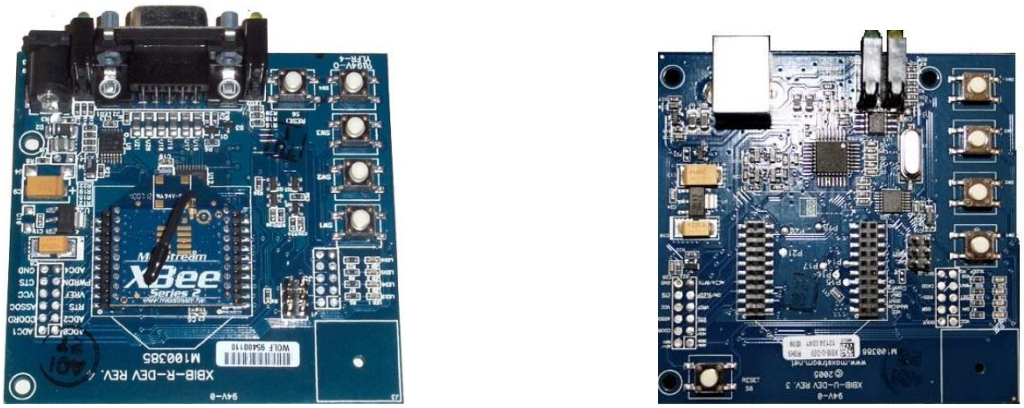
Los módulos XBee son soluciones embebidas de radiofrecuencia que implementando el protocolo IEEE 802.15.4, permiten conectividad inalámbrica entre módulos. Con la implementación del estándar IEEE 802.15.4 se consigue tener un protocolo rápido para la comunicación en redes punto a punto y en redes *peer-to-peer*. Están diseñados para tener una alta confiabilidad en las transmisiones, requiriendo poco tiempo entre transmisiones y tiempos de comunicación predecibles.

Ilustración 2.31: Módulo XBee

Las principales características del módulo XBee son:

- No necesita configuración para empezar a funcionar como transmisor RF.
- Configuración de pines común a varios módulos de radio frecuencia.
- Alta tasa de transferencia de datos de hasta 250 Kbps hacia dispositivos finales.
- Transmite en la banda de 2.4GHz.
- Soporta modo hibernación para ahorrar energía. (*Sleep mode*)

Los módulos XBee y XBee-PRO son compatibles pin a pin, y su mayor diferencia es el precio y su poder de transmisión.



(a) (b)

Ilustración 2.32: (a) - Placa de desarrollo RS232, (b) - Placa de desarrollo USB

2.5 Bibliografía y referencias

[1] Wireless Sensor Network: Yang Yu,Victor K Prasanna,Bhaskar Krishnamachari. "Information processing and routing in wireless sensor networks" Ed: World Scientific, (2006)

[2] Definición del estándar IEEE 802.15.4: *"IEEE 802.15.4 - Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)"*, revisión 2006.

[3] *Carrier sense multiple access with collision avoidance*, Acceso múltiple por detección de portadora, evitando colisiones. Disponible on-line en: Pablo Brenner, "A Technical Tutorial on the IEE 802.11 Protocol", BreezeCOM, 1997. Disponible on-line en: http://www.sss-mag.com/pdf/802_11tut.pdf

[4] Open *System Interconnection*, Interconexión de sistemas abiertos, Disponible on-line en: http://standards.iso.org/ittf/PubliclyAvailableStandards/s020269_ISO_IEC_74981 1994(E).zip

[5] Media Access Control. Es una subcapa de la capa Enlace de datos del modelo OSI y su función principal es gestionar el acceso al canal de comunicación y el direccionamiento. Más información: A. Tanenbaum, "Computer networks" 4ª Ed. Prentice Hall, Año 2003.Cap 4.

[6] Zigbee protocol, Disponible online en: http://www.zigbee.org/Products/-DownloadZigBeeTechnicalDocuments.aspx

[7] GTS , Definición en: "On the IEEE 802.15.4 MAC Layer Attacks: GTS Attack", CAP3.

[8] IPv6 over *Low power Wireless Personal Area Networks,* IPv6 sobre Redes de área personal inalámbricas de bajo consumo. Disponible on-line en: http://datatracker.ietf-.org/wg/6lowpan/charter/

[9] *Internet protocol version 6*. Disponible on-line en: http://tools.ietf.org/html/rfc2460

[10] TinyOS Operating System. Disponible on-line en: http://www.tinyos.net

[11] Contiki Operation System. Disponible on-line en: http://www.sics.se/contiki/

[12] LiteOs. Sistema operativo de tiempo real y código abierto con base UNIX diseñador para redes de sensores inalámbricos. Disponible on-line en: http://www.liteos.net/

[13] Pequeño sistema operativo de tiempo real diseñado para CPUs de 8 bits. Más información on-line en: http://www.ethernut.de/en/firmware/nutos.html

[14] Nano-RK es un sistema operativo de tiempo real para redes de sensores inalámbricos desarrollado en la Universidad Carnegie Mellon. Disponible on-line en: http://www.nanork.org/.

[15] Mantis, Sistema operativo con soporte de múltiples hilos desarrollado por la Universidad de Colorado y escrito en C, para redes de sensores inalámbricos. Más información disponible on-line: http://mantisos.org

[16] Lenguaje de programación NesC: Más información on-line: http://nescc.sourceforge.net/

[17] Virtual Network Computing. Protocolo de control del ordenador a distancia. Disponible definición del protocolo on-line en: http://www.realvnc.com/docs/rfbproto.pdf

[18] *Telecomunication Network*. Protocolo de acceso a máquinas conectadas en red para manejarlas remotamente. Disponible definición del protocolo on-line en: http://www.faqs.org/rfcs/rfc854.html

[19] Proyecto financiado por el DARPA y desarrollado por la universidad de Berkeley para construir motes con sensores de de tamaño inferior a 1 mm cuadrado: Disponible on-line en: http://robotics.eecs.berkeley.edu/~pister/SmartDust/

[20] Compañía especializa en la fabricación de chips para la conversión de diversos interfaces a USB. Disponible más información on-line en: http://www.ftdichip.com/

[21] Empresa líder en la fabricación de sensores de humedad y flujo en líquidos. Listado de producto disponible on-line en: http://www.sensirion.com/

[22] Empresa desarrolladora de WaspMote. Disponible on-line: http://www.libelium.com/

[23] Empresa creadora y distribuidora del kit de desarrollo ZigBit. Disponible on-line: http://www.meshnetics.com/

[24] Kit de desarrollo de la casa Microchip para redes 802.11.4. Disponible on-line: http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId= 1406 &d DocName=en021925

[25] Kit de desarrollo para redes 802.11.4 de Texas Instruments. Disponible on-line en: www.ti.com/eZ430-RF2480.

[26] Entorno de desarrollo para construir y depurar aplicaciones basadas en el microcontrolador MSP430. Descarga libre online en : http://focus.ti.com/docs/-toolsw/folders/print/iar-kickstart.html

[27] Implementación de bajo consumo del protocolo Zigbee realizada por Texas Instruments. Más información online en: http://focus.ti.com/docs/prod/-folders/print/cc2480a1.html

[28] Kit de desarrollo lanzado por la empresa Zolertia. Encontramos más información online en: http://www.zolertia.com/products/Z1-starter-kit

[29] Sistemas de código abierto, se pueden distribuir libremente y se tiene acceso a su código fuente. Más información online en: http://www.opensource.org/docs/osd

[30] *Puerto de* Entrada/Salida con capacidades "Plug and Play" para la conexión de sensores y actuadores simples de la compañía Zolertia. Más información online en: http://www.zolertia.com/

[31] Empresa distribuidora de Xbee y Xbee Pro tras la adquisición de Maxstream. Disponible on-line: http://www.digi.com

Capítulo 3

Descripción del Kit de Desarrollo XBee

3.1 Introducción

En el capítulo anterior se han estudiado las redes inalámbricas de sensores y algunos de los kits de desarrollo disponibles para llevar a cabo el desarrollo de aplicaciones con esta tecnología. Entre los kits de desarrollo descritos, se ha elegido el kit de desarrollo del módulo XBee de Digi.

El XBee *Starter Development kit* tiene como principal característica su simplicidad, ya que normalmente los kits de desarrollo están formados por un micro-controlador y un módulo de radio con diversos periféricos como sensores, entradas salidas, puertos serie o USB, etc. Sin embargo, este kit de desarrollo está compuesto por el modulo de comunicaciones XBee y la placa de interface, en la cual únicamente se encuentran LEDs, pulsadores, el chip de conversión de niveles de tensión TTL a la salida de puerto del interface USB o RS232 (mediante los chips FT232BM [1] y MAX232 [2] respectivamente) y los adaptadores de la alimentación.

Como se observa en la placa de interface mostrada en la Ilustración 3.1, el XBee no tiene ningún micro-controlador externo. Esto es así porque el módulo XBee lleva

incorporado un pequeño micro-controlador de 8 bits. Concretamente en el XBee este micro-controlador es el MC9S08GT60 del fabricante Freescale [3].

Además, este módulo de tan solo 20 pines va insertado en un zócalo sobre la placa de interface, lo que hace que sea muy sencillo construir placas de interface adaptadas a las necesidades en un espacio muy reducido (ver detalle en la Ilustración 3.1), además de poder programarlo en un placa que podría llevar puerto serie y luego colocarlos sobre una placa de diseño propio que sólo tuviese el módulo, la alimentación y los sensores o actuadores.

Cabe destacar que el XBee es capaz de funcionar como transmisor RF, únicamente con la conexión de 4 pines (Din(Rx), Dout (Tx), GND, VCC).

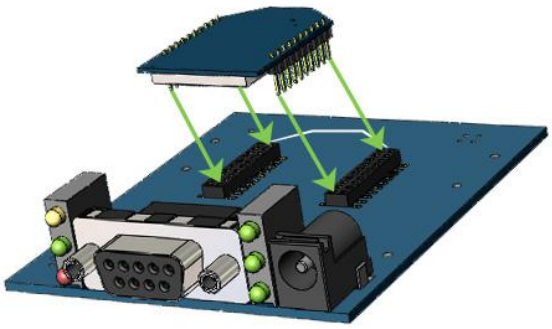


Ilustración 3.1: Vista esquemática de módulo XBee y placa de interface

Por otra parte, este módulo proporciona una gran cantidad de entradas salidas, tanto digitales como analógicas, que se pueden usar fácilmente para realizar una gran variedad de desarrollos.

Los módulos XBee han sido diseñados para cumplir con el estándar IEEE 802.15.4 y soportar las redes inalámbricas de sensores de bajo coste, baja potencia de transmisión, bajo consumo, pero ofreciendo fiabilidad en las comunicaciones entre los dispositivos, siendo un módulo muy completo, con muchas posibilidades y reducidas dimensiones. Interiormente los XBee llevan montado el chip de radio MC13193 [4] del fabricante Freescale. En la Ilustración 3.2 se puede observar su forma y distribución de los pines de los módulos XBee y XBee Pro, cuya mayor diferencia es la diferente potencia de transmisión de sus radios, y por tanto, el mayor rango de alcance del módulo XBee Pro.

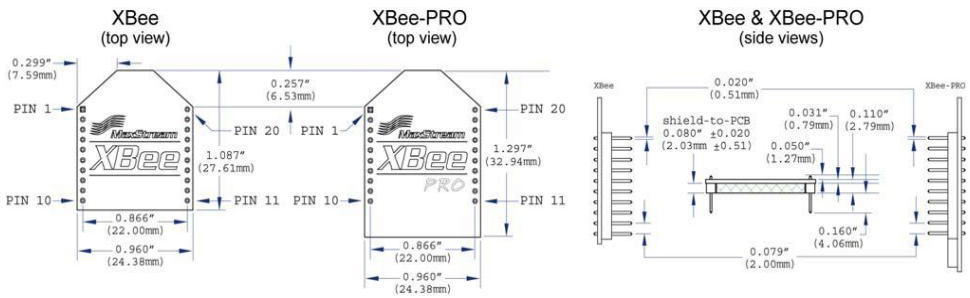


Ilustración 3.2: Dimensiones del módulo XBee

En la Tabla 3.1 se observa de forma resumida las principales características de los módulos XBee que funcionan en la banda libre de 2.4 GHz.

Especificaciones		XBee	XBee-Pro
Rendimiento	Alcance en ambientes interiores / zonas urbanas	Hasta 30 m	Hasta 100 m
	Alcance de RF en línea de Visión para ambientes exteriores	Hasta 100 m	Hasta 1200 m
	Potencia de salida de Transmisión	1 mW (0 dB)	60mW (18 dB), 100mW EIRP
	Régimen RF de datos	250.000 bps	250.000 bps
	Sensibilidad del receptor	-92 dBm(1% PER)	-100 dBm (1% PER)
Rendimientos de potencia	Rango de alimentación	2.8-3.4 V	2.8-3.4 V
	Corriente de transmisión	45 mA a 3.3 V	270 mA a 3.3V
	Corriente de recepción	50 mA a 3.3 V	55 mA a 3.3 V
	Corriente Power-Down	<10 uA	<10 uA
Información general	Frecuencia	ISM 2.4 GHz	ISM 2.4 GHz
	Dimensiones	2.438 cm x 2.761 cm	2.438 cm x 3.292 cm
	Tª de operación	-40 a 85º (industrial)	-40 a 85º (industrial)
	Precio aproximado	19 $	32 $
	Opciones de antena	Conector U.FL, Antena en Chip o antena de cable.	Conector U.FL, Antena en Chip o antena de cable.
Trabajo en Red	Topologías permitidas en la red	Punto a Punto, Punto a Multipunto, entre pares y Mesh	Punto a Punto, Punto a Multipunto, entre pares y Mesh
	Número de canales	16 canales de secuencia directa	12 canales de secuencia directa
	Capas de filtración de la red	PAN ID y direcciones 64 bits	PAN ID y direcciones 64 bits
Redes y seguridad	Reintentos y confirmaciones DSSS (*Direct Sequence Spread Spectrum*) Tienen alrededor de 65000 direcciones disponible de red Direccionamiento Fuente/Sumidero Comunicaciones de Broadcast (todos dentro de la red) o a un destinatario.		
Otras características	Soporte de entradas analógicas y digitales I/O Line parsing (Cableado virtual de entrada salidas) Facíl de usar. Preconfigurado para usarse como transmisor RF por el puerto serie. El software de configuración (X-CTU) es gratuito. Soporte de comandos AT y API para la configuración de los módulos.		

Tabla 3.1: Características principales de los módulos XBee

3.2 Hardware

Este módulo, a pesar de tener unas reducidas dimensiones (25mm x 27mm), proporciona hasta siete entradas analógicas, nueve entradas/ocho salidas digitales y dos salidas PWM, todo ello en 20 pines. Para ello, permite configurar por software la función de los pines, para que actúen de un modo u otro. En la Tabla 3.2 se observa la función de cada pin y sus posibles configuraciones, estos pines se empiezan a contar desde la izquierda y en el sentido contrario de las agujas del reloj. En la Ilustración 3.3 se ve claramente la disposición de los pines del módulo XBee.

Pin	Nombre	Dirección	Descripción
1	VCC	-	Tensión de alimentación
2	DOUT	Salida	UART Salida de datos
3	*DIN* / $\overline{CONFIG}$	Entrada	UART Entrada de datos
4	DO8*	Salida	Salida digital 8
5	RESET	Entrada	Modo reset
6	PWM0/RSSI	Salida	Salida PWM0 / Indicador de señal recibida
7	PWM1	Salida	Salida PWM 1
8	[reservado]	-	No conectado
9	$\overline{DTR}$ / *Sleep _ RQ* / *DI*8	Entrada	Pin de control de línea de sleep o Entrada digital 8
10	GND	-	Tierra
11	AD4/DIO4	Entrada/Salida	Entrada analógica 4 o entrada/salida digital 4
12	$\overline{CTS}$ / *DIO*7	Entrada/Salida	Control de flujo o entrada/salida digital 7
13	*ON* / $\overline{SLEEP}$	Salida	Indicador modulo de estado
14	VREF	Entrada	Referencia de voltaje para entradas analógicas / digitales
15	ASSOCIATE/ AD5 / DIO5	Entrada/Salida	Indicador asociado / entrada analógica 5 o entrada /salidas digital 5
16	RTS / AD6 / DIO6	Entrada/Salida	Solicitud de envío de control de flujo / entrada analógica 6 o entrada/salida digital 6
17	AD3 / DIO3	Entrada/Salida	Entrada analógica 3 o entrada/salida digital 3
18	AD2 / DIO2	Entrada/Salida	Entrada analógica 2 o entrada/salida digital 2
19	AD1 / DIO1	Entrada/Salida	Entrada analógica 1 o entrada/salida digital 1
20	AD0 / DIO0	Entrada/Salida	Entrada analógica 0 o entrada/salida digital 0

Tabla 3.2: Pines de los módulos XBee y XBee Pro

De los pines descritos anteriormente, para tener un módulo operativo las conexiones mínimas necesarias solo son cuatro: VCC, GND, DOUT y DIN. Si se quiere poder actualizar su firmware o cambiar sus parámetros ha de conectarse también RTS y DTR. Con esta configuración el módulo ya está listo para funcionar.

Este módulo también dispone de resistencias de *pull-up* [5] en sus entradas digitales y son configurables mediante software en la configuración del XBee (usando el comando PR, *Pull-up Resistor enable*). Es de destacar también la presencia de una resistencia interna de 50kΩ en el pin de RESET, lo que permite tener este pin sin conectar cuando no se necesita reiniciar el módulo.

Ilustración 3.3: Esquema de pines

3.2.1 Características Eléctricas del XBee

En la Tabla 3.3 se describen las principales características eléctricas y consideraciones a tener en cuenta para la alimentación principal del módulo, así como los diferentes consumos del módulo y las características de tensiones máximas y mínimas de las entradas salidas.

Descripción	**Mínimo**	**Máximo**
Tensión de alimentación	2,8 V	3,4 V
Tensión mínima nivel alto entradas (V_{IH})	0,7 * VCC	
Tensión máxima nivel bajo entradas (V_{IL})		0,35 * VCC
Tensión máxima nivel bajo salidas (V_{OL})		0,5 V
Tensión mínima nivel alto salidas (V_{OH})	VCC – 0,5	
Consumo en transmisión por radio (TX)	45 mA	
Consumo en recepción via radio. (RX)	50 mA	
Consumo en modo hibernación	< 1 uA	
Tensión referencia para ADC. (V_{ref})	2.08 V	VCC
Resolución ADC	(V_{ref} – V_{GND}) / 1024 V / punto	

Tabla 3.3: Características eléctricas del XBee

3.2.2 Modos de Operación Módulos XBee

Los módulos XBee están preparados para funcionar en cinco modos de operación o estados, según la tarea que han de realizar. Estos estados son los siguientes: transmitiendo, recibiendo, entrada de comandos, modo *sleep,* que es el modo de bajo consumo de energía o también llamado estado suspendido y modo de reposo (*Idle Mode*). El modo *Idle* es el modo principal al que siempre se retorna desde los demás, tal y como se observa en el diagrama de modos y transiciones mostrado en la Ilustración 3.4.

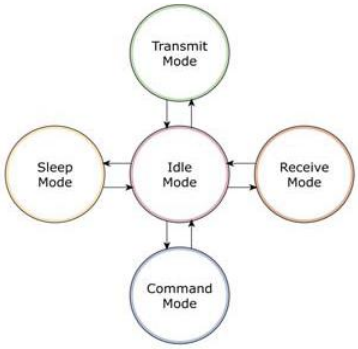


Ilustración 3.4: Modos de operación del XBee

Las tareas y funciones del módulo XBee en cada uno de sus estados de funcionamiento son los siguientes:

Idle mode: Modo reposo, el módulo estará en este estado siempre que este activo, no estando ni recibiendo ni transmitiendo datos, ni en modo comando. Desde este modo, se puede cambiar a cualquiera de los demás modos, y es al que se vuelve siempre que termina cualquiera de los otros modos. En este modo el consumo es de 50 mA.

Sleep mode: El modo suspendido permite entrar al módulo en un estado de bajo consumo cuando no es usado. En este modo no se envía ni recibe nada por la radio. El módulo puede entrar a este modo por recibir una señal en el pin 9 (*Sleep RQ*), petición de bajo consumo o por software, si se ha cumplido el tiempo del parámetro ST (*Time before Sleep*). En este modo el consumo es < 50 μA.

Estos módulos disponen de tres modos de suspensión diferentes que se pueden configurar con el parámetro SM. Estos modos son:

Hibernación (SM=1): Es el modo de menor consumo, menor de 10 μA. Este menor consumo tiene como contra el mayor tiempo que tarda en pasar al estado activo. Sólo se puede pasar de este modo a estado activo desactivando la entrada al pin *Sleep_RQ*.

Suspendido (SM=2): En este modo es más rápido su vuelta a modo activo, pero su consumo es unas 5 veces más elevado que el anterior, alrededor de 50 μA.

Modo cíclico (SM0=4,5): El módulo entra y sale del modo suspendido según el tiempo definido en sus parámetros (ST- *Time before Sleep* y SP- *Cyclic Sleep Period*). En este modo tiene el mismo consumo y el tiempo de vuelta a modo activo que en el caso anterior, pues en los periodos de inactividad está en el mismo estado suspendido.

Nota: El consumo en los modos suspendidos depende mucho de la tensión de alimentación del módulo, a mayores tensiones, por encima de los 3V, los consumos en modo *sleep* aumentan exponencialmente con el aumento de la tensión de entrada.

Receive mode: El módulo está recibiendo datos. El consumo es de 50mA.

Transmit mode: El modulo está transmitiendo datos. El consumo es de 45mA.

Los paquetes pueden ser transmitidos por dos métodos: directo o indirecto.

Transmisión directa: Los paquetes son enviados inmediatamente a su dirección de destino. Este modo es usado en redes que no disponen de coordinador, ejemplo, una red donde todos los nodos son dispositivos finales. En una red con coordinador se usa este tipo de transmisión cuando un dispositivo final hace una petición de datos al coordinador. Este último puede decidir usar este tipo de transmisión para enviar los datos al dispositivo final siempre que ambos tengan configurado el parámetro ST (*Time before sleep* – tiempo antes de ir a modo suspendido) con valores similares.

Transmisión indirecta: El paquete es puesto a la espera de ser enviado y no es enviado hasta que el módulo de destino le realiza una petición de datos. Este tipo de transmisión sólo es posible en redes con coordinador. Normalmente se usa para asegurar la recepción de paquetes por los dispositivos finales que pasan parte de su tiempo en modo suspendido.

Procedimiento para el envío

Antes de cada envío de paquetes, se realiza un escaneo de la energía del canal para ver el nivel de ocupación del canal, para que si este está ocupado esperarse a que quede libre antes de intentar enviar el paquete. Esta comprobación del canal es llamada CCA (*Clear Channel Asesessment*), y consiste en detectar la energía del canal, por el que se va a enviar el paquete y comparar su energía con el valor del parámetro CA en el XBee. Si el valor de la energía del canal es inferior a este, se retrasa el envío hasta que se descongestione el canal y su nivel de energía baje por debajo del parámetro CA.

Además, para aumentar la fiabilidad en las comunicaciones se puede exigir el envío de un reconocimiento (*acknowledgement*) por parte del receptor. Si el reconocimiento no es recibido por el módulo que envía el paquete, este reenviará dicho paquete hasta tres veces

si no recibe el *ack* (*acknowledgement*). En caso de no recibirlo en ninguna de las veces, se avisará de un error de *ack* en el módulo emisor.

Command mode: Modo comando, utilizado para la configuración del módulo. En este modo se pueden leer y escribir los parámetros de configuración del módulo. En este estado los caracteres recibidos son interpretados como comandos. Los módulos soportan dos tipos de modo comando diferente, el modo AT y el modo API.

Modo AT: En este modo es posible configurar el módulo XBee mediante un terminal conectado por el puerto serie a la UART del XBee, enviando caracteres ASCII. Este modo de configuración se describirá posteriormente.

El modo API, Sirve para leer o escribir comandos en el módulo, además de ser el medio en que se envían las lecturas y escrituras de los pines de entrada/salida del módulo.

Este modo tiene la peculiaridad que permite la configuración remota del módulo, y es el utilizado para el uso de los módulos desde aplicaciones informáticas, dado que permite el envió y recepción de paquetes dirigidos a un destinatario concreto y diferente sin tener que cambiar el parámetros de destino en el módulo, simplemente añadiendo al paquete a enviar la dirección del destinatario (DL, DH – *Destination Address Low, High*). Este modo de funcionamiento y configuración se describirá posteriormente.

3.2.3 Funciones Especiales del XBee

***DIO Change detect* (Detección de cambio en entradas digitales)**

Si se tiene el XBee configurado para leer el estado de sus entradas, la función *DIO Change dectect* permite enviar un paquete con los valores de las entradas ante un cambio en alguna de estas, lo que evita estar enviando paquetes con la misma información, ahorrando así batería, al disminuir el número de envíos sin perder información para sistemas donde los cambios en las variables de medida son esporádicos. Para que la detección del cambio de señal en alguna de sus entradas digitales sea reconocido el módulo debe de estar en modo activo (no detectaría el cambio en modo suspendido). Si el módulo esta en modo suspendido sólo puede detectar cambios en el pin DI8 (*Sleep / wake*), que es el pin usado para pasar al módulo al estado activo mediante hardware.

Cuando un paquete se envía por detección de cambio de valor en sus entradas digitales, no se envían los valores de las señales analógicas, solo se envían los valores de las entradas digitales. Además, si el módulo está configurado para esperar tener varios paquetes antes de enviarlos, cuando detecte el cambio en sus pines enviará los paquetes que tenga en el buffer junto con el nuevo paquete, sin esperar a tener el número especificado por su parámetro IT (*Samples before transmit*).

IO Line passing

El *IO Line passing* consiste en crear un cable virtual entre las entradas de un módulo XBee y las salidas de otro. Cuando un paquete de datos que contiene información de entradas/salidas es recibido, si el módulo está configurado para realizar *IO Line Passing*, actualiza el valor de sus salidas (tanto PWM como salidas digitales) con los valores del paquete recibido.

Este emparejamiento de entradas y salidas ha de hacerse con las mismas entradas/salidas en ambos módulos, es decir, si el módulo A esta configurado para transmitir sus entradas, si lee en el AD0, entonces el módulo B, configurado para actualizar sus salidas, lo hace con el PWM0, el DI2 actualiza el DO2, y así respectivamente, hasta el ADC/DI7, puesto que el DI8/*Wake*/*Sleep* no puede ser usado para este cometido.

Se puede hacer que el módulo que recibe los paquetes para actualizar su salida sólo escuche a un solo módulo que le enviará la información de sus entradas, o que lo haga de cualquier módulo de la red. Se configura mediante el parámetro IA (I/O *Input Address*). Si se quiere que cualquier módulo de su red que este apropiadamente configurado, pueda cambiar el estado de sus salidas, se ha de configurar este parámetro a IA = 0xFFFF. En caso de que se quiera que sólo actualice sus salidas con la información de un módulo en particular, se le asigna la dirección de este módulo. Esta restricción de escucha de un solo módulo sólo se tiene en cuenta para la actualización de sus salidas, no para la recepción de otro tipo de paquetes.

Las salidas que pasan del estado inactivo al estado activo a través del *IO Line Passing* tienen una duración determinada que se fija mediante los temporizadores T1 – T6 (D1-D6 *Output Timer*) para las salidas digitales de D0 a D6 y PT (PWM *output Timer*) para las salidas de modulación de ancho de pulso PWM0 y PWM1, para estas salidas el valor de PT es compartido para ambas. Una vez expirado este tiempo las salidas vuelven a su estado inactivo.

El tiempo máximo configurable para estos temporizadores es de 0xFF x 100 ms, es decir como máximo se pueden mantener estas salidas durante 25,5 segundos. Si se quiere que las entradas estén activas más tiempo de los 25,5 segundos permitidos por los temporizadores, se tendrá que fijar el periodo de envío de los paquetes en el módulo emisor, a un valor más pequeño que el tiempo máximo permitido para los temporizadores, para que así se refresquen las salidas antes de que expiren los temporizadores.

Estos temporizadores se resetean cada vez que llega un paquete con información válida. En caso de que no llegue ningún paquete válido en el tiempo configurado, las salidas pasarían a su estado no activo.

3.2.4 *Tipos de Redes*

Estos módulos siguiendo los modelos de redes del estándar IEE 802.15.4 tienen la posibilidad de formar dos tipos de redes: Redes Peer to Peer y redes con coordinador sin sincronización entre nodos.

Redes *peer to peer*

Este tipo de red viene configurado por defecto en los módulos XBee. En este tipo de red no es necesaria la presencia de un coordinador y todos los módulos están configurados como dispositivos finales. Todos se comunican con todos los demás módulos en el alcance de su radio. En la Ilustración 3.5 se observa la estructura básica de esta topología. En este tipo de red los módulos permanecen sincronizados haciendo cada uno de ellos las funciones de maestro/esclavo.

Esta arquitectura de red está construida siguiendo la homóloga del estándar IEEE 802.15.4. Con esta red se consigue tener tiempos de creación y configuración pequeños. Lo que la hace idónea para un gran número de aplicaciones.

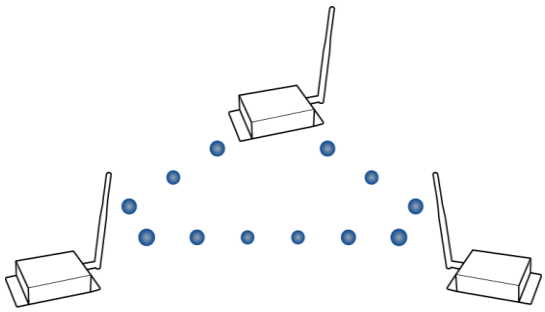


Ilustración 3.5: Red peer to peer

Redes con coordinador, sin sincronización entre nodos (non beacon)

En este tipo de red se establecen los roles de maestro/ esclavo, donde un nodo será el coordinador y los demás serán dispositivos finales, implementando así la topología en estrella definida por el estándar IEEE 802.15.4 y mostrada en la Ilustración 3.6.

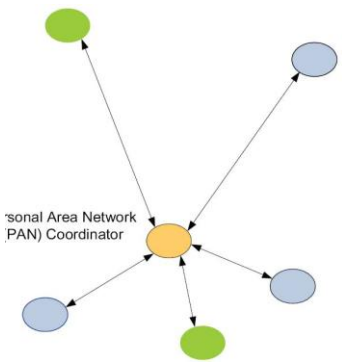


Ilustración 3.6: Red en Estrella con Coordinador

El coordinador es el encargado de mantener el sincronismo en la red y de controlar los mecanismos de creación de la red. Puede ser configurado para trabajar con transmisiones directas o indirectas, dependiendo del valor del parámetro SP (*Cyclic Sleep Period*). Si los dispositivos finales van a pasar gran parte de su tiempo en modo activo se usará la transmisión directa. En cambio si se quiere preservar las baterías en los dispositivos finales, se pueden pasar estos a modo suspendido parte del tiempo y entonces convendrá usar el modo indirecto de transmisión. En el caso de usar transmisión indirecta, se ha de tener en cuenta que coincidan los valores de SP (*Cyclic Sleep Period*) y ST (*Time before Sleep*) en el coordinador, así como en los dispositivos finales.

Para que los dispositivos finales formen parte de una red y el coordinador los reconozca como parte de su red, se tiene que producir el proceso de asociación. Este proceso consiste en establecer una relación de pertenencia de un dispositivo final a una red gobernada por un coordinador, a estas redes se les conoce como PAN (*Personal Area Network).* En las PAN cada dispositivo tiene un identificador único dentro de la red, el PAN ID, el cual no debe repetirse en redes adyacentes para evitar problemas de comunicación entre diferentes redes.

Para llevar a cabo la asociación un módulo necesita tener configurado uno de los siguientes tres parámetros sobre la red a la que quiere conectarse: el PAN ID, la dirección del coordinador o el canal en el que funciona. Con cualquiera de estos tres parámetros el módulo puede establecer contacto con el coordinador de la red para asociarse. Los parámetros que no se conocen han de configurarse para ser auto-detectados en el momento de iniciar el proceso de asociación.

3.3 Modos

Los módulos XBee disponen de diferentes modos de trabajo: modo reemplazo de puerto serie, modo comando y modo API (*Application Programming Interface*), lo que les permite adaptarse fácilmente a un gran número de aplicaciones. Estos modos se seleccionan por medio de los parámetros de configuración. Por defecto viene configurado para funcionar como reemplazo de una comunicación serie. En este modo simplemente lo que recibe por su UART lo transmite por RF hasta otro XBee. Este módulo al recibirlo lo transmitirá por su UART hasta el dispositivo que tenga conectado a su interface. El modo comando será el usado para programar los parámetros de configuración del XBee y el modo API es una combinación de los anteriores modos, permitiendo mediante una serie de tramas que se enviarán o recibirán por la UART, realizar envíos y recepciones de datos entre módulos vía radiofrecuencia, así como el envío de comandos de configuración, que pueden ir destinados al módulo directamente conectado a la UART o a otro módulo en el

alcance de su radio, además de ser el modo necesario para poder transmitir la información recogida por las entradas analógicas y digitales.

3.3.1 Modo Transparente

El modo transparente viene pre-configurado en los módulos XBee. Al operar en este modo, los módulos actúan como sustitución del puerto serie. Todos los datos recibidos a través de el pin DI (pin3 XBee - *Data In*) del XBee pasan a la cola de transmisión por RF (*Radio Frequency*). Cuando se reciben los datos por RF, se envían por el pin DO (pin 4 XBee, *Data Out*), que es el pin de salida se la UART del XBee.

Este tipo de comunicación puede realizarse con cualquier UART con una lógica y tensión compatible a la de la tecnología CMOS, o bien a través de un convertidor desde niveles del estándar RS-232 de otros dispositivos (por ejemplo con el puerto serie estándar de un PC, a través del IC MAX2322). En la Ilustración 3.7, se muestra un ejemplo de este modo, así como las señales utilizadas para realizar el reemplazo de una comunicación serie por otra inalámbrica usando los XBee. En este modo, el XBee creará una trama con los datos recibidos y los enviara por radio. El otro módulo al recibirlos decodificará dicha trama y enviará por su UART solamente los datos de la trama recibida, de ahí el nombre de comunicación transparente, pues el usuario no se preocupa de cómo se envían los datos.

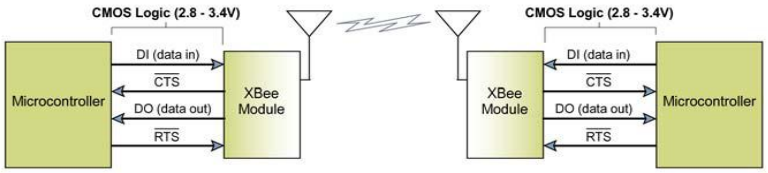


Ilustración 3.7: Comunicaciones Xbee

Para que el XBee trabaje en modo transparente es necesaria la configuración de los parámetros indicados en Tabla 3.4:

CH	Canal	*C*
MY	Dirección propia	***0***
DH	Dirección destino Alta	***0***
DL	Dirección destino Baja	***0***
A1	Association	0
ID	PAN Ide	***1234***
CE	Coordinador enable	0
AP	API Enable	0

Tabla 3.4: Parámetros de configuración modo transparente

Los valores remarcados en cursiva de la Tabla 3.4 pueden ser cualquier valor dentro de sus rangos, pero han de ser los mismos en todos los módulos de la red. Para que los datos sean transmitidos a todos los módulos.

Principios de funcionamiento

El módulo UART es el encargado de la temporización y la comprobación de la paridad, que se necesitan para las comunicaciones de datos. Las comunicaciones serie dependen de que las dos UART tengan una configuración compatible (velocidad en baudios, la paridad, bits de inicio, bits de parada, bits de datos), es decir, que estos parámetros estén configurados a los mismos valores en ambos lados de la comunicación.

En el módulo, los datos entran a través del pin DI de la UART (pin 3) como una serie de señales asíncronas. Cada byte de datos consiste en un bit de inicio (nivel bajo), 8 bits de datos (bit menos significativo primero) y un bit de parada (nivel alto). Un ejemplo de este formato de señales de la UART se observa en la Ilustración 3.8. El número de los bits de datos y el de los bits de parada es definido cuando se establece la comunicación, al igual que la velocidad de transferencia que marca el tiempo entre bits.

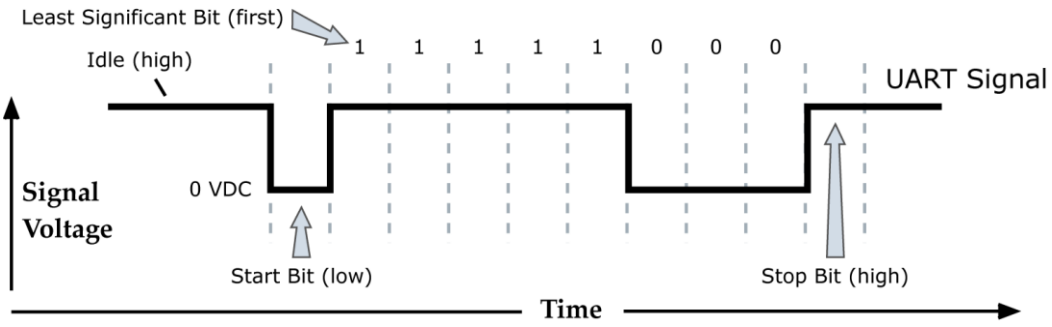


Ilustración 3.8: Transmisión serie de un byte conteniendo el valor 31 (0x1F16)

A continuación se describen los elementos principales que intervienen en la transferencia serie del XBee hacia la radio y viceversa. En la Ilustración 3.9 se muestran los elementos que componen este traspaso de datos desde la radio hacia la UART y viceversa. Se muestra como la antena puede sólo estar recibiendo o enviando, pero no ambos a la vez, ya que se selecciona su comportamiento mediante el conmutador de la radio.

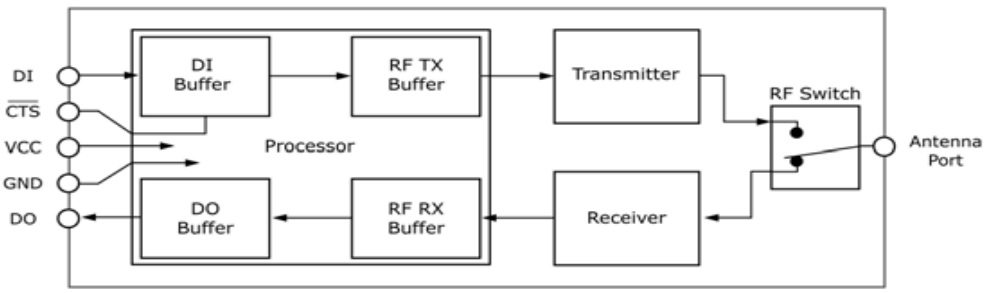


Ilustración 3.9: Diagrama de componentes de la UART

Buffer DI (datos de entrada)

Cuando los datos serie entran al módulo de RF a través del pin DI (pin 3), se almacenan en el buffer DI hasta que se pueden procesar. Los datos esperan en el buffer de entrada DI hasta que ocurre una de las 3 causas siguientes:

- No se reciben caracteres serie en el tiempo determinado por el parámetro RO (*Packetization Timeout*). Si RO = 0, el empaquetado y envío comienza cuando se recibe un solo carácter.

- Se llega a 100 caracteres, que es el número máximo de caracteres que caben en un paquete de RF.

- Se recibe el modo de secuencia de comandos GT + CC + GT. Cualquier carácter que estuviera almacenado en el búfer DI antes de la secuencia, será transmitido.

Si el módulo no puede transmitir de inmediato (por ejemplo, si ya está recibiendo datos de RF), los datos se almacenan en el búfer de DI. Si el buffer DI se llena y sigue sin poder transmitir los datos vía RF, entonces se activan en la UART las señales de control de flujo (hardware o software) con el fin de evitar desbordamiento del buffer y la pérdida de datos.

Control de flujo

Para evitar el desbordamiento del buffer DI, se puede usar la señal de control de flujo (CTS – *Clear To Send*). Cuando al buffer DI le faltan 17 bytes para estar completo, por defecto, el módulo pone a nivel alto la señal CTS para que el dispositivo que está transmitiendo detenga el envío de datos. CTS vuelve al nivel bajo cuando el buffer DI tiene más de 34 bytes de memoria disponibles.

Hay escenarios donde no es necesario un control de flujo porque no se pueden producir desbordamientos del buffer. Algunos de estos casos son:

1. Enviando mensajes más pequeños que el tamaño del buffer de DI. Es decir menores a 100 bytes.

2. Haciendo que la velocidad de recepción de datos de la interfaz serie sea menor que la tasa de envío de datos a través de RF.

Por otro lado, los datos de DI se podrían desbordar y por consiguiente sería necesario tener un control de flujo en el siguiente caso:

3. Si el módulo está continuamente recibiendo datos a través de la radio, cualquier dato que le llegara por la UART se almacenaría en el buffer de entrada DI a la espera de que el módulo de radio quede libre, pero si dado la carga de recepción de transmisiones no queda liberada, el buffer DI se desbordaría y se perdería la información que siguiera llegando a él.

Buffer DO (datos de salida)

Cuando se reciben datos de RF, los datos entran al buffer DO y son enviados al puerto serie del otro dispositivo receptor. Una vez que se llena el buffer de salida DO porque no se pueden enviar los datos al destinatario, si se reciben más datos por RF se pierden. Si la señal RTS (*Request to Send*) está habilitada para el control de flujo, los datos no serán enviados al buffer DO mientras RTS esté activa.

Los dos casos en los que el buffer DO puede llegar al desbordamiento son:

1. Si la velocidad de transmisión de datos de RF es mayor la tasa de datos de la interfaz del módulo, el módulo recibirá datos desde la radio para transmitir al dispositivo receptor más rápido de lo que él puede enviarlos por el interfaz serie.

2. Si el receptor no permite que el módulo transmita los datos del buffer DO, debido a las señales de control de flujo hardware o software.

3.3.2 Modo de Comandos AT

El modo de comandos AT es utilizado para la lectura y escritura de los parámetros de configuración del módulo XBee a través del puerto serie. En este modo, cuando los caracteres son recibidos en el puerto serie, no son enviados a través de la radio, sino que son interpretados como comandos de configuración del módulo. Para operar en este modo es necesario introducir una secuencia de caracteres para activarlo en el XBee.

Para entrar en modo comando se ha de seguir el siguiente protocolo:

1- No enviar ningún carácter durante un segundo, antes del paso 2.

2- Envío de tres caracteres "+++" en menos de un segundo.

3 - No enviar ningún carácter en el segundo posterior a la introducción de los tres caracteres anteriores.

Los tiempos de espera antes y después, de los tres caracteres se pueden cambiar mediante el comando GT (*Guard Time*), así como definir cuál será el carácter de entrada "+" mediante el parámetro CC (*Command Sequence Caracter*) por otro cualquiera.

Envío de parámetros AT

Para enviar comandos AT, una vez se tiene el XBee conectado mediante el puerto serie y se ha cambiado al modo de comandos según el procedimiento anteriormente descrito, se tiene que seguir la sintaxis mostrada en la Ilustración 3.10, para el envío de los comandos.

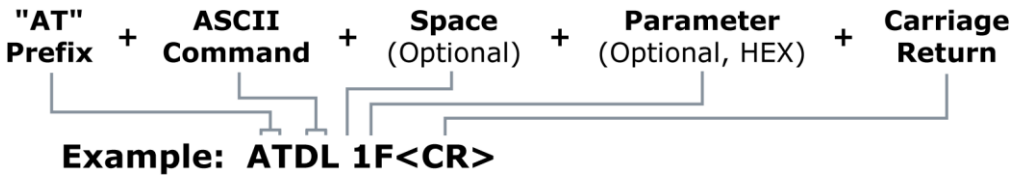


Ilustración 3.10: Sintaxis de comando AT

Para la escritura y ejecución de comandos se ha enviar el prefijo AT, el comando en ASCII, el parámetro que se desea asignar a ese comando y se ha de pulsar INTRO, que equivale a enviar el carácter de retorno de carro o 'CR'.

Tras la correcta recepción del comando por el módulo, este devuelve un mensaje de OK, o en caso de error un mensaje de ERROR. Es importante recordar que si se quiere que los comandos enviados no se pierdan tras un reinicio del módulo, se ha de enviar el comando WR (*Write*) de escritura para que los parámetros enviados se queden grabados en la memoria no volátil.

Para la lectura del parámetro se sigue el mismo formato visto anteriormente pero se omite el parámetro y el módulo devuelve el resultado del comando ejecutado. Entre el comando y el parámetro se puede dejar un espacio o escribir el parámetro seguidamente. De ambos modos lo interpreta igual.

Salir del modo de comandos AT

Para salir del modo comando se tienen dos opciones:

- Enviar el comando ATCN (*Exit command mode*) seguido de INTRO.
- Esperar sin enviar ningún carácter el tiempo especificado en el parámetro CT (*Command mode Timeout*).

En el anexo IV se adjunta una tabla con todos los posibles comandos AT que acepta el módulo XBee y su funcionalidad. Esta información también se puede encontrar en el "XBee *Product Manual*" páginas 27-34 [6].

3.3.3 Modo API

El modo de operación API (*Application Programming Interface*) es una alternativa al modo Transparente. En este modo se pueden crear redes e incluso que el módulo interactúe con un *host* (por ejemplo un PC) a través de comandos.

La API proporciona diferentes alternativas para configurar los módulos, así como diferentes tipos de ruteo de los datos. Una aplicación puede enviar datos a diferentes módulos sin tener que cambiar la dirección de destino mediante la configuración del módulo, ya que cada *frame* (trama) enviada contiene un campo con la dirección de destino. Además, el modo API permite recibir confirmación de las transmisiones realizadas. Así se sabe si un paquete se ha entregado correctamente o no. En cuanto a la recepción de paquetes, este modo permite identificar al módulo que envió el paquete, puesto que el *frame* también incluye la dirección del nodo transmisor.

En el modo API, todos los datos que entran y salen del módulo deben de seguir una estructura de interface (los datos son transmitidos usando *frames* en un orden específico). La API especifica como los comandos, las respuesta de comandos y los *frames* de estados o eventos del módulo son enviados y recibidos, usando los *frames* de datos de la UART. Aquí hay una doble encapsulación, pues hay una sintaxis simple para los comandos que son enviados por la UART y luego la propia de la API, que va contenida en el *Payload* de esta.

Cuando el módulo esta en modo API, todos los caracteres recibidos por la UART son tramas (*frames*) de comando API. Al igual que los enviados han de seguir la sintaxis de comandos API. Por tanto, si se quiere obtener información de ellos se han de procesar. Toda la información que llega al módulo en este modo, si no cumple las reglas de formación o falla en la comprobación de su *checksum* es directamente descartada sin producirse ninguna notificación.

Es únicamente en este modo donde se pueden realizar lecturas y escrituras de las entradas y salidas de los módulos en modo automático, como por ejemplo que el módulo envíe la lectura de sus sensores conectados periódicamente. Cuando se configura en el módulo la lectura de entradas periódicamente, el módulo únicamente envía los resultados en una trama API.

Manualmente se pueden usar tres comandos AT, teniendo el módulo conectado localmente por puerto serie. IS (*Force Sample*) que permite conocer el estado de las entradas digitales y analógicas, IO (*Digital Output level*) que permite fijar los valores de las salidas digitales, y M0 o M1, (PWM1,2 *Output Level*) que permite leer y fijar los valores de las salidas de PWM0 y PWM1. Este proceso se debe repetir en cada iteración.

Existen 5 tipos de *frames,* 2 para la transmisión (1 y 2) y 3 para la recepción (3-5) de datos y comandos:

1) *Frame* de datos para transferir por RF.

2) *Frame* de comando, donde va contenido alguno de los comandos AT.

3) *Frame* de recepción de datos.

4) *Frame* de respuesta de un comando.

5) *Frame* de notificación de eventos, como puede ser, *reset, asocciate, dissasociate*, etc.

Para que el módulo trabaje en modo API se ha de configurar el parámetro AP, del siguiente modo:

- AP=0 (Por defecto): Modo de operación transparente.
- AP =1 : Modo API. (Modo sin caracteres de escape)
- AP = 2: Modo API con caracteres de escape.

En este trabajo se usa sólo el modo de trabajo sin caracteres de escape.

El modo API con caracteres de escape se usa para evitar conflictos con ciertos bytes que podrían ser confundidos por la UART al decodificar. Estos bytes son, 0x7E (*Delimitador frame*), 0x7D (Escape), 0x11 (XON) y 0x13 (XOFF). Para evitarlo se introduce en la posición de la trama donde esta contenido alguno de estos bytes, dos nuevos bytes, el primero el byte de escape, 0x7D seguido de otro byte calculado como una XOR (Operación lógica OR exclusiva) entre el byte a escapar y 0x20.

Estructura de la trama en modo API

En el modo API las tramas o *frames* se envían a través de la UART, por tanto tienen que seguir la especificación de una trama UART, que es la mostrada en la Ilustración 3.11 .

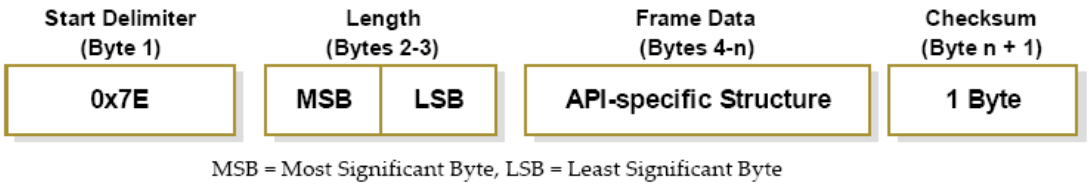


Ilustración 3.11: Estructura general de una trama UART

El significado de cada uno de los campos de la estructura es el siguiente:

***Start Delimeter*:** Delimitador de comienzo de trama UART, es siempre 0x7E.

Length: Indica el tamaño en bytes del campo "*Frame Data*".

Frame data: Es donde va contenida la información, así como los distintos tipos de estructuras especificas de los comandos de la API.

Checksum: Sirve para comprobar la integridad de la información enviada o recibida. Se calcula, sin incluir los delimitadores, ni la longitud, sumando todos los bytes y manteniendo solo los 8 bits de la parte más baja del resultado, restándole 0xFF. El resultado de esto es el byte del *checksum*. Para su verificación, es necesario sumar los bytes de los datos y si es correcto debe de ser igual a 0xFF.

La estructura de la trama API específica, donde irán contenidos los comandos o los datos a transmitir, como se puede observar en la Ilustración 3.12, está incluida en el campo *Frame data* de la trama UART.

Dentro del campo *Frame Data* se encuentra la información específica del comando API. Este campo siempre se compone de dos subcampos, el "*cmdID*", con una longitud fija de un byte, que es el identificador de tipo de comando API que se envía o recibe y el campo "*cmdData*" de longitud variable, donde va contenido el comando del API. Los comandos posibles y su estructura se detallan a continuación.

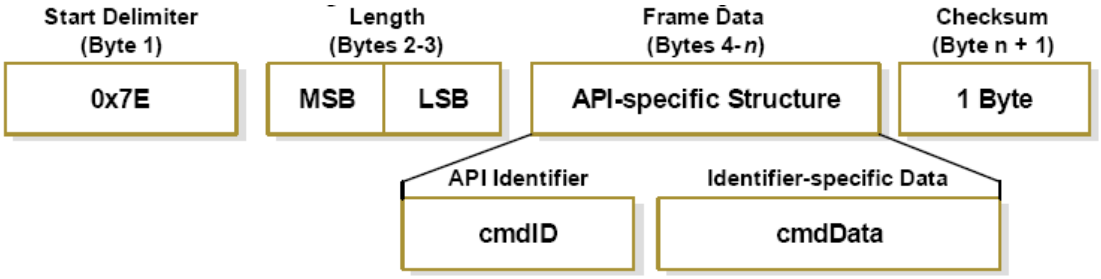


Ilustración 3.12: Estructura de la trama API especifica, contenida en una trama UART

Comandos

La parte más importante del modo API son los comandos que se pueden enviar y recibir y que se compone de 13 tipos de comandos de tipo API diferentes. Mediante estos comandos es posible conocer el estado del módulo, mandar y recibir comandos AT, enviar y recibir paquetes de datos, recibir lecturas de las entradas digitales y analógicas, así como enviar y recibir comandos AT remotos.

Como se indicó anteriormente, cada uno de estos tipos de comandos posee un identificador único. El identificador es una parte importante del comando, ya que cada

tipo de comando tiene su propio formato y el identificador permite seleccionar el modo correcto de extraer la información, una vez que se reciben las tramas a través de la UART, en el programa de tratamiento. Además, esta información es utilizada por los módulos XBee para la correcta decodificación de las tramas que reciben cuando trabajan en este modo.

La Tabla 3.5 resume los tipos de comandos que es posible enviar/recibir en el modo API y sus correspondientes identificadores

Identificador API	Tipo de comando
0x8A	Estado del modem.
0x08	Comando AT.
0x09	Comando AT encolado. (Se envía pero no se ejecuta hasta recibir el AC *(Apply Changes* – Aplicar cambios)
0x88	Respuesta de comando AT.
0x17	Comando AT Remoto.
0x97	Respuesta de comando remoto.
0x00	Transmitir datos. (Dirección 64 bits)
0x01	Transmitir datos. (Dirección 16 bits)
0x89	Resultado de la transmisión datos.
0x80	Recepción de datos. (Dirección de 64 bits)
0x81	Recepción de datos. (Dirección de 16 bits)
0x82	Recepción de datos conteniendo información de entradas digitales / analógicas (Dirección de 64 bits). Usa el formato de 0x80.
0x83	Recepción de datos conteniendo información de entradas digitales / analógicas (Dirección de 16 bits). Usa el formato de 0x81.

Tabla 3.5: Tipos e Identificadores de comandos API

Entradas/Salidas

Uno de los principales motivos de haber elegido este módulo es debido a la gran cantidad de entradas/salidas digitales y analógicas que posee. En concreto proporciona hasta 9 entradas, 8 salidas digitales, o en caso de las analógicas hasta un máximo de 6 entradas analógicas, aunque no todas pueden actuar a la misma vez, dado que en el módulo un mismo pin puede ser entrada, salida digital o entrada analógica. Entonces dependiendo de su configuración hará una función u otra.

También existen dos pines que se pueden configurar como salida de modulación de anchura de pulso (PWM, *Pulse Width Modulation*), permitiendo de esta manera disponer de hasta 2 salidas analógicas.

En la Tabla 3.6 se muestran las posibilidades de configuración de cada pin.

Pin	Descripción	Entrada / Salida digital	Entrada Analógica	Salida PWM
6	PWM0			X
7	PWM1			X
15 – 20	DIO6/AD6 – DIO/AD0	X	X	
11	DIO4/AD4	X	X	
12	DIO7	X		
9	DI8	X (Solo entrada digital)		

Tabla 3.6: Posibilidades de configuración de los pines del XBee

Para la configuración de la función de los pines digitales y de entrada analógica se usa el comando: ATDn (Donde "n" es el número de entrada/salida), este comando tiene las siguientes posibilidades de configuración, respetando las limitaciones de función de cada pin indicadas en la Tabla 3.6.

- ATDn = 2; Conversión analógica digital. (Entrada analógica)
- ATDn = 3; Entrada digital.
- ATDn = 4; Salida digital nivel bajo.
- ATDn = 5; Salida digital nivel alto.

Para la configuración de los pines de salida PWM, se utiliza el comando P0 (*PWM0 Configuration*) para configurar el pin 6. El comando P1 *(PWM1 Configuration*) se utiliza para configurar el pin 7 del módulo XBee, para que sea salida PWM, indicador de la intensidad del último paquete recibido (RSSI) o dejarlo deshabilitado.

Los valores de configuración de este comando para realizar las tareas descritas anteriormente son los siguientes, donde x es 0 ó 1 en función del pin de PWM a configurar:

- Px = 0; Pin desactivado
- Px = 1; Indica el RSSI, la intensidad de la señal del último paquete recibido.
- Px = 2; Salida PWM

Es muy importante para las entradas analógicas fijar el valor de VREF, cuando se realiza el conexionado del módulo, este valor se puede fijar a una tensión máxima de (0,7 * VCC), que será el máximo admisible para cada entrada analógica o digital.

La resolución de los convertidores AD (analógicos-digitales) es de 10 bits, así para 0 V en la entrada tendrá un valor de 0 digital y para valores de VREF o superiores en la entrada tendrá el valor máximo de 3FF ($2^{\wedge 10}$ -1= $3FF_{16}=1023_{10}$).

A modo de ejemplo, si se tiene una tensión de alimentación VCC = 3,3 V, y una VREF configurada a 2 V, para entradas analógicas. Para una entrada AD1 = 1 V se obtendrá en la parte digital 1FF, la mitad del fondo de escala.

Cada vez que un módulo envía información sobre el estado de sus entradas, lo realiza mediante el formato de una trama de E/S. El formato de esta trama consta de la cabecera mostrada en la Ilustración 3.13 y el cuerpo que es mostrado en la Ilustración 3.14. La información que aparece en la cabecera es una máscara indicando las entradas que están activas, ya sean digitales o analógicas, y en el cuerpo se encuentra la información de las entradas digitales, y tras estas está la información de las entradas analógicas, si el módulo las tiene configuradas para lectura.

La trama de entradas digitales y analógicas remotas está compuesta por la cabecera y el cuerpo, y va encapsulada en el campo de datos (*cmdData*) de una trama API para la recepción de datos, pero con un identificador propio en el campo "*cmdId"* que indica que la información a tratar es de entradas digitales/analógicas. Este identificador es 0x82 para paquetes direccionados con dirección de 64 bits y de 0x83 para paquetes direccionados con dirección corta, de 16 bits.

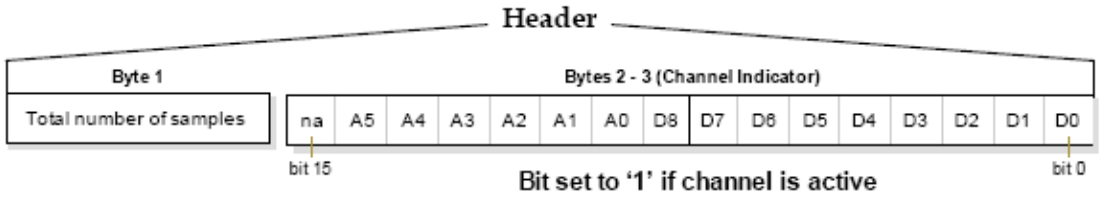


Ilustración 3.13: Cabecera de una trama de E/S

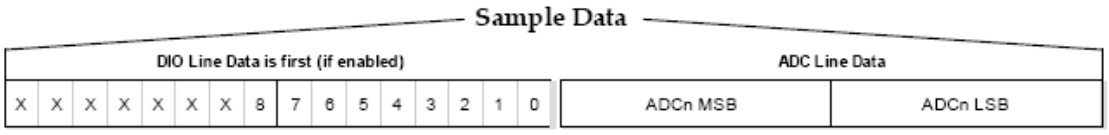


Ilustración 3.14: Cuerpo de una trama de E/S

El módulo XBee tiene funciones en su sistema de entradas y salidas, que son de mucha utilidad para la realización de proyectos de monitorización y control, como es la posibilidad de programar que pase de modo de bajo consumo a modo activo en periodos de tiempo prefijados y realice un muestreo de sus entradas, además de comprobar si tiene mensajes pendientes de recibir o enviar.

Los parámetros básicos para la configuración de lectura de las entradas son:

- SM = 4 o 5, Define el tipo de ciclo de *Sleep* (Modo suspendido).
- IR= 0; Periodo de muestreo.
- IT, Nº muestras a recoger antes de proceder al envío por la radio.
- ST, Tiempo de espera antes de volver a modo suspendido.

El módulo muestrea una vez si tiene IT=1. Si IT tiene otro valor y el IR esta ajustado, el módulo no volverá a modo *sleep* hasta que el número de muestras necesarias para proceder al envió (IT) se cumpla.

3.4 Configuración

La lectura y modificación de los parámetros del módulo se realiza a través de su interfaz RS-232, bien mediante introducción de comandos AT o de comandos API. Desde "Digi International" [7] el fabricante de estos módulos se pone a disposición de los usuarios el programa de configuración X-CTU [8], en el que se puede probar el correcto funcionamiento de las radios, actualizar el *firmaware* de los módulos y configurarlos rápidamente.

Utilización:

Lo primero que se debe de hacer es establecer comunicación desde el PC hasta el módulo a través del puerto serie. Lo más sencillo es tener el XBee montado en una de las placas de desarrollo que vienen en el kit, se puede utilizar tanto la placa RS-232 como la USB, con la salvedad que esta última no necesita ser alimentada externamente pues toma su alimentación del puerto USB. Después de conectarlos, se elige el puerto serie donde está conectado el módulo.

La configuración del puerto serie en los módulos por defecto es la mostrada en la Tabla 3.7:

Velocidad:	9600 baudios
Control flujo:	Sin control de flujo
Pariedad:	Ninguna
Bit Datos:	8
Bit Stop:	1

Tabla 3.7: Configuración por defecto módulos XBee

3.4.1 Software X-CTU

En la primera pantalla del Software de configuración X-CTU, mostrada en la Ilustración 3.15 se configuran los parámetros de conexión con el módulo.

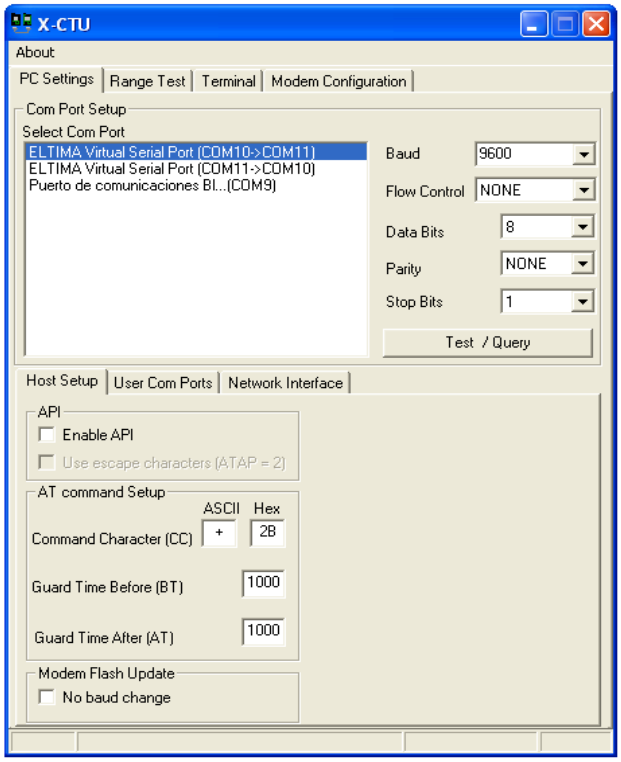


Ilustración 3.15: Software de configuración X-CTU

Presionando sobre el botón Test/Query se comprueba si se ha establecido una correcta conexión con el módulo.

El software X-CTU también permite probar el alcance de la radio. Para probar el alcance de la radio se necesitan al menos dos módulos, y se conectan de la forma mostrada en el Ilustración 3.16.

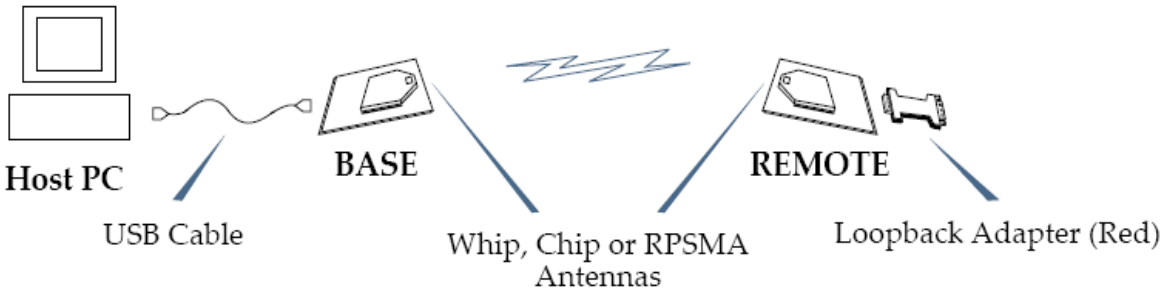


Ilustración 3.16: Configuración de test para módulos XBee

Como se muestra en la Ilustración 3.16 se usa una de las placas de interface para conectar el XBee al PC. La otra con puerto RS-232 y con alimentación por batería se le conecta un conector de *loopback*, que hará que todo lo que se reciba por el puerto serie sea enviado otra vez por el mismo. Ambos XBee deben de estar activos en modo Transparente[9]. (Este modo viene preseleccionado por defecto)

En el PC se ha de ejecutar el software X-CTU y comprobar que se tiene conexión con el módulo conectado. Una vez que se tiene conexión se puede ir a la pestaña de la aplicación de "*Range Test"*, presionando "*START*", comenzará a enviar paquetes de datos, a través del módulo conectado al PC, y si el otro módulo esta dentro del alcance se recibirá los mismos paquetes que se enviaron y la intensidad de la señal al ser recibidos (RSSI, *Receive Signal Strength Indication*). Una vez realizado el test, aparecerá una ventana similar a la de la Ilustración 3.17. Así se puede usar este test para saber cuál es el límite del alcance en la zona donde se van a instalar.

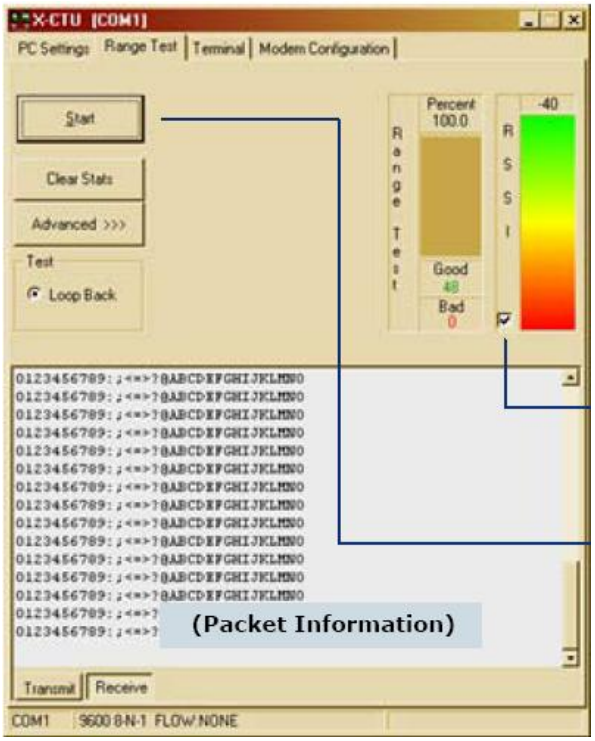


Ilustración 3.17: Pantalla del *"Range Test"*

La pantalla de terminal serie mostrada en la Ilustración 3.18 es muy útil para entrar en modo comando e introducir comandos aislados de configuración, visualizar o modificar los existentes. Para configurar todo el módulo es preferible hacerlo con la pestaña de configuración del módem. En esta pantalla de terminal lo que se teclea se enviará al módulo y aparecerá en color azul, lo que se recibe del módulo aparece en color rojo.

Pantalla terminal

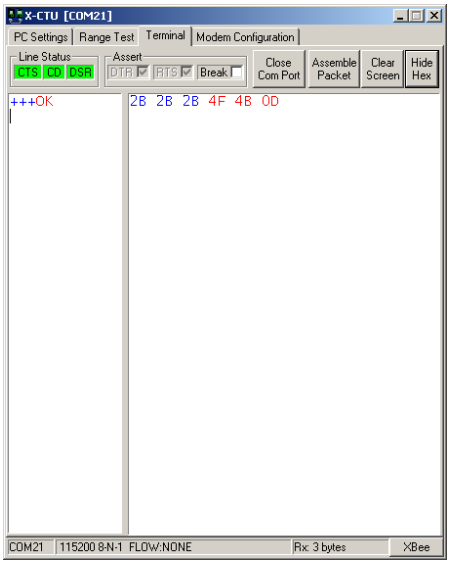


Ilustración 3.18: Terminal X-CTU

Para enviar comandos AT, se sebe de escribir mediante la estructura AT + Comando + Parámetros, vista en el apartado 3.3.2. A continuación se muestra un ejemplo para conocer el canal que tiene configurado para la radio el XBee.

Enviado: ATCH + <CR> // Consulta al módulo el canal que tiene configurados
// para las comunicaciones de radio.

Recibido: > C // El módulo responde indicando que está configurado
// para transmitir / recibir por el canal C.

El botón *Hide/Show Hex* hace que aparezca un divisor de pantalla que permite tener la traducción hexadecimal de los caracteres enviados o recibidos, lo cual resulta de especial interés cuando se trabaja con el modo API, ya que este modo se trabaja sobre bytes y si sólo se tiene su información ASCII no se ve realmente la información que contienen, por ejemplo direcciones de envío, cabeceras, longitudes, las entradas analógicas y binarias, etc.

Desde el botón "*Assamby packet*" es posible escribir una secuencia de comandos o bytes y luego enviarlos todos a la misma vez. Esto resulta útil por ejemplo en el modo API para preparar un paquete entero y enviarlo. Es importante no generar un paquete de más de 100 bytes ya que desbordaría el buffer de salida del XBee y se perderían datos.

Configurando todo el modem:

Configurar de una vez todos los valores del módem es posible desde la pestaña "*Modem Configuration*", mostrada en la Ilustración 3.19, así como leerlos y grabarlos en un fichero o leerlos de un fichero almacenado en el PC y pasarlos al XBee.

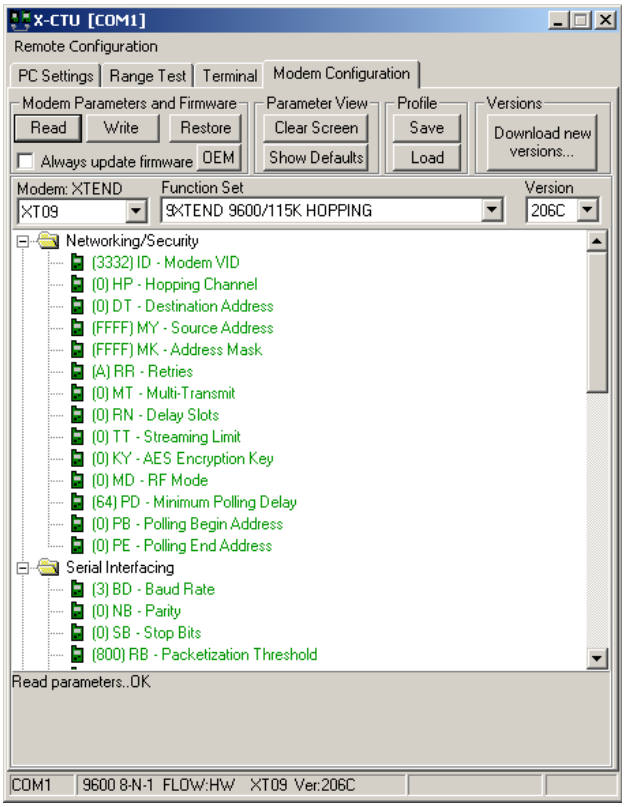


Ilustración 3.19: Configuración general del módem

También es posible en esta pestaña actualizar el firmware de los módulos, señalando la casilla de "*Always update firmware*" cuando se hace un *write* o *restore* de los comandos, el firmware del módulo se actualiza a la versión elegida en la lista desplegable "*Version*" de la derecha.

Es importante tener en cuenta que se ha de tener la misma versión de firmware en todos los módulos con los que se trabaje. En general lo importante es que el coordinador y los dispositivos finales tengan la misma versión de firmware, en este trabajo al probar diferentes configuraciones se tendrán todos los módulos con la misma versión de firmware en concreto la versión 0x10ED.

3.5 Placas de Desarrollo

3.5.1 Introducción

El *XBee Starter Development kit*, está formado por dos placas de desarrollo, una placa con interface RS-232, denominada XBIB-R-DEV y otra con interface USB denominada XBIB-U-DEV.

Ambas placas tienen cuatro pulsadores, para permitir cambiar el estado de los pines configurados como entradas digitales, un pulsador de *reset* para reiniciar el módulo, un

adaptador de alimentación con regulación a las tensiones de trabajo del XBee. Este último permite a la placa tener un amplio rango de alimentación, desde 6V hasta 20V. También disponen de cinco LEDs que permiten visualizar el estado de pines configurados como salidas digitales, tres LEDs, situados a la izquierda del interface de conexión con el PC, que indican la actividad de los pines Tx y Rx del XBee, visualizan la actividad de los pines de la UART, y un tercer LED de color rojo, que monitoriza el pin *Associate* del XBee indicando así cuando el XBee se encuentra asociado con un Coordinador. Otros tres LEDs situados en el lado derecho del interface de conexión con el PC, indican la cantidad de salida analógica que se tiene en el pin PWM0, que por defecto viene configurado para dar como salida el RSSI del último paquete recibido.

En el centro de la placa se localizan dos zócalos de 10 pines cada uno, para insertar el módulo XBee, el paso de pines es de 2 mm. En los laterales de la placa se encuentran los huecos y los agujeros, para poder insertar zócalos de paso estándar 2,54mm, que dan acceso a cada uno de los pines del XBee.

Situado en la parte lateral donde se sitúa el XBee, estas placas proporcionan 3 *jumpers*, donde se alojará el conector para la programación de los módulos. Los XBee de este trabajo no tienen partes programables por este método, pero se usan para la programación del *Firmware* del micro-controlador interno de los XBee.

La característica que diferencia estas placas es el tipo de interface de conexión con el PC. La placa denominada XBIB-R-DEV, mostrada en la Ilustración 3.20, posee un conector DB-9 actuando como puerto serie, para poder interactuar directamente con un puerto serie de un PC.

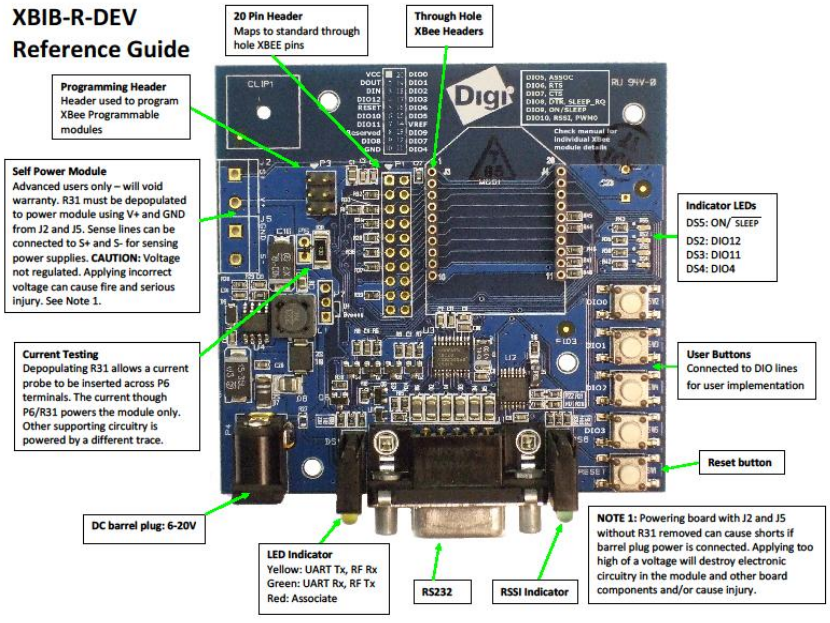


Ilustración 3.20: Placa de desarrollo XBIB-R-DEV

Esto es posible gracias al transductor RS232, que permite adaptar los niveles de tensión de trabajo de los puertos series de los PCs, con los niveles TTL de 3V que tiene el XBee. En esta placa, concretamente se utiliza el chip SP3203EEY del fabricante Sipex [10].

La placa de desarrollo XBIB-U-DEV, mostrada en la Ilustración 3.21, tiene como interface para la conexión con el PC un conector USB de tipo B. Este conector tiene dos funciones, hacer de interfaz con el PC y proporcionar la alimentación necesaria de la placa, no siendo necesarias las baterías si está conectado al PC. Cuando se conecta al PC con el USB, la placa automáticamente desconecta la alimentación desde el conector de alimentación y pasa a alimentarse desde el USB. La función de comunicación a través de cualquier puerto USB del PC, se tiene por medio de la emulación de un puerto serie, a través de USB. Para el correcto funcionamiento de este mecanismo es imprescindible la instalación de los drivers proporcionados por el fabricante, la primera vez que se conecta la placa de desarrollo al PC. El encargado de gestionar la conexión USB en la placa y pasar las señales del USB a señales RS232 con niveles de la UART del XBee, es el chip FT232BM del fabricante FTDI.

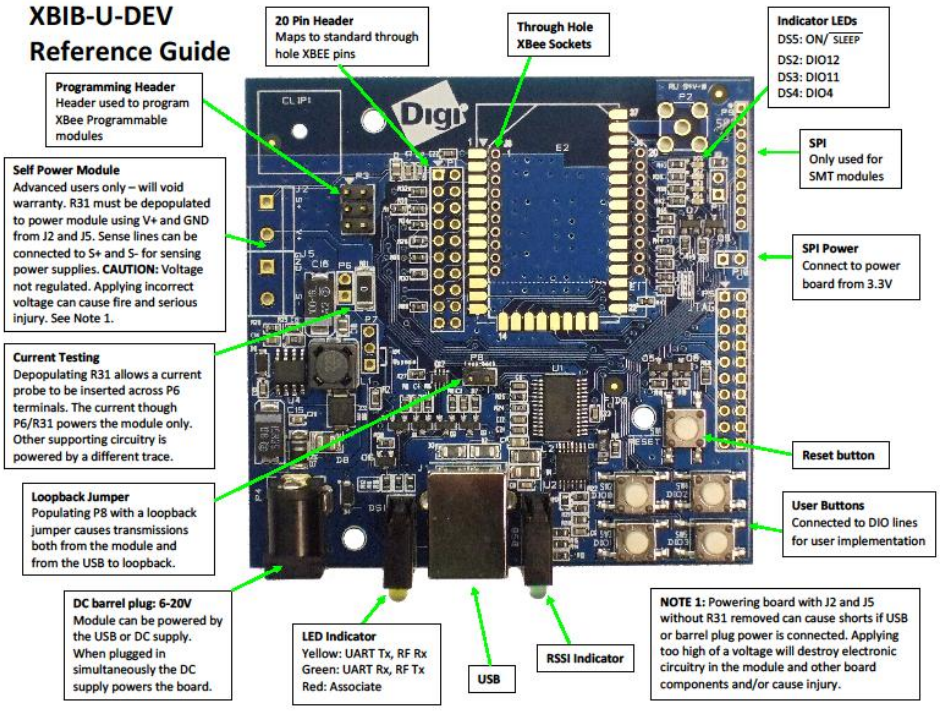


Ilustración 3.21: Placa de desarrollo XBIB-U-DEV

En la siguiente página se muestra la Ilustración 3.22 donde está el esquemático de la placa de desarrollo XBIB-R-DEV y la Ilustración 3.23, que contiene el diseño esquemático de la placa de desarrollo XBID-U-DEV.

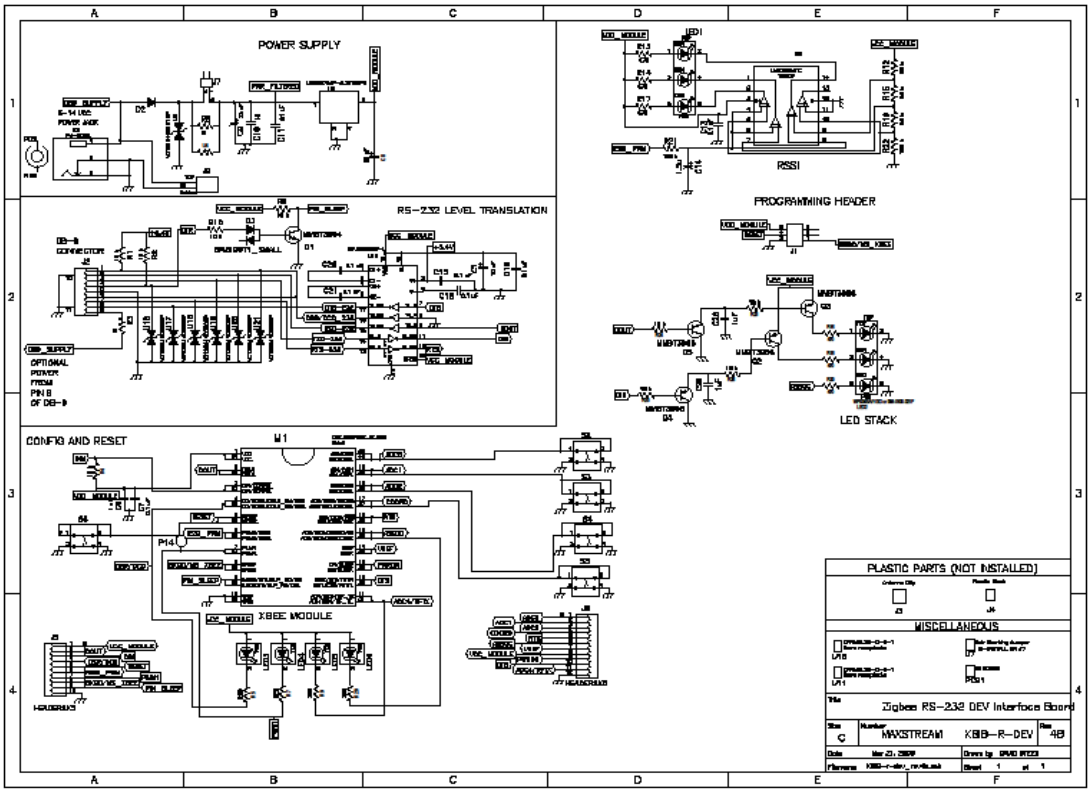


Ilustración 3.22: Esquemático XBIB-R-DEV

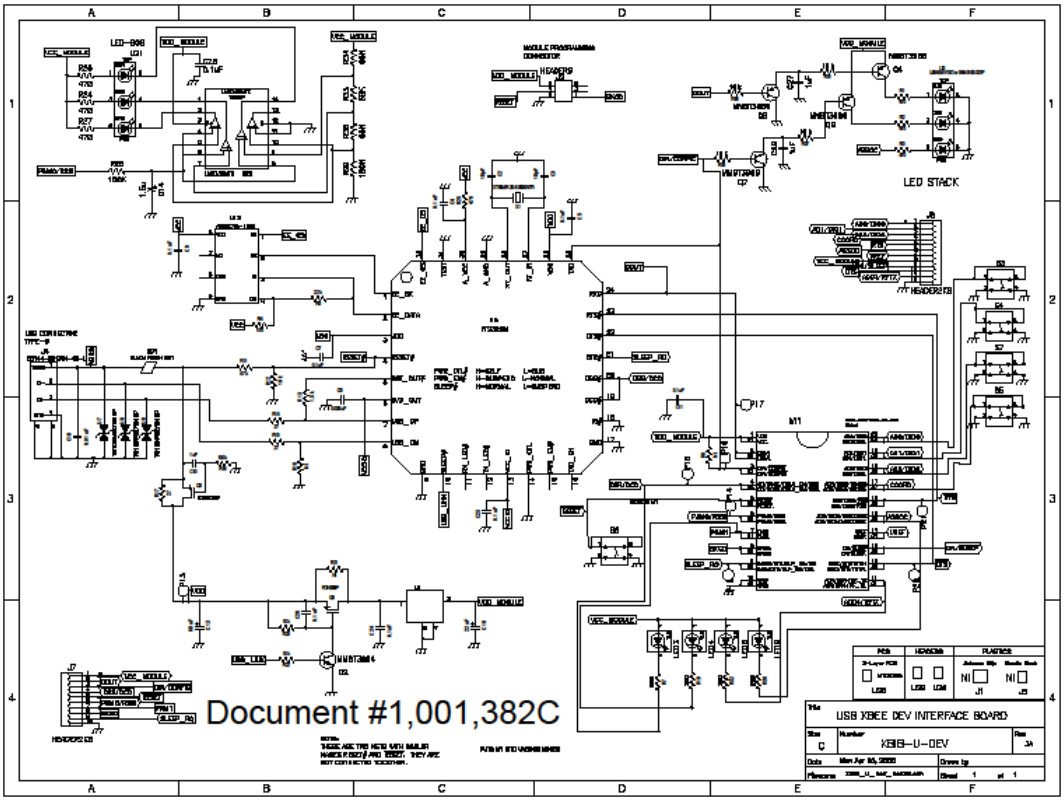


Ilustración 3.23: Esquemático XBIB-U-DEV

3.6 Conclusiones

Para implementar la WSN de este PFC se seleccionó el módulo de radio XBee por lo completo y reducido que es, ya que incorpora en el mismo chip las funciones de radio y el microcontrolador, así como un gran número de entradas analógicas y digitales, lo que lo hacen perfecto para actuar como nodo sensor en una red. Además, su configuración y comunicación con el PC resulta muy sencilla al llevar incorporado una UART, con lo que la comunicación con el PC es inmediata. También es importante destacar que se pueden realizar cambios en módulos remotamente, lo que facilita las labores de mantenimiento al no tener que desplazar el módulo al laboratorio o taller para realizar cambios en sus parametrización.

Las placas de desarrollo a usar con el XBee tienen un pequeño tamaño pues el XBee ya incorpora muchas de las funcionalidades que normalmente se incorporan en la placa de montaje.

En este capítulo se ha descrito el hardware del módulo XBee. También se han descrito sus modos de funcionamiento, así como la secuencia necesaria para establecer uno de estos modos, y el funcionamiento en cada uno de ellos. Se ha descrito el hardware de las placas de desarrollo que serán el elemento sobre el que, en el capítulo siguiente se realizarán las pruebas de los modos de funcionamiento vistos en este capítulo y se configuraran los módulos XBee para trabajar en diferentes supuestos, y comprobar las características que ofrecen estos completos módulos de radio XBee, para conformar WSN.

3.7 Bibliografía

[1] Chip adaptador USB – Serie del fabricante Ftdichip. Disponible on-line en: www.ftdichip.com/Products/FT232BM.htm

[2] Adaptador de niveles lógicos entre puertos RS232 y UARTs con tecnología TTL. Disponible on-line en: www.maxim-ic.com/datasheet/index.mvp/id/1798

[3] Se puede encontrar más información sobre este microcontrolador on-line en la página del fabricante: http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=S08GT&fsrch=1&sr=10

[4] Chip de radio que se basa en el estándar IEEE 802.15.4 para sus transmisiones. Más información on-line en: http://www.freescale.com/webapp/sps/site/-prod_summary.jsp?code=MC13193

[5] Resistencia de polarización, permite dejar pines al aire y que no adquieran valores aleatorios.

[6] XBee®/XBee-PRO® OEM RF Modules. "Product Manual v1.xCx - 802.15.4 Protocol Digi International", 2008. Disponible on-line en: http://ftp1.digi.com/support/-documentation/90000982_B.pdf

[7] Digi international, Making Wireless M2M Easy: Web online: http://www.digi.com/products/wireless-wired-embedded-solutions/zigbeerfmodules-/point-multipoint-rfmodules/xbee-series1module.jsp#overview

[8] Programa para la configuración de los módulos XBee. Se puede descargar libremente desde el sitio on-line del fabricante: http://www.digi.com/support/kbase-/kbaseresultdetl.jsp?kb=125

[9] Robert Faludi,"Building wireless sensor networks", Ed: O'Reilly, 2011. CAP2- "*Up and Running*".

[10] Transceiver de señales RS-232 a niveles TTL de 3 Vdc. Del fabricante Sipex Datasheet de este chip disponible on-line: http://www.datasheetcatalog.org/datasheet/sipex/-SP3203E.pdf

Capítulo 4

Modos de Operación del XBee

4.1 Introducción

El módulo XBee viene configurado por defecto para operar como un simple reemplazo inalámbrico del cableado serie. No obstante, este módulo tiene muchas más funciones, que son más útiles desde el punto de vista del control inalámbrico de aplicaciones, las cuales se desarrollaran a lo largo de este capítulo.

4.2 Configuraciones XBee

En este apartado se describen las diferentes pruebas realizadas con los módulos XBee para diferentes configuraciones. La configuración del módulo se puede modificar introduciendo los parámetros necesarios mediante comandos AT o modificando los comandos en la ventana de configuración, y posteriormente grabándolos en el módulo XBee por el puerto serie.

4.2.1 Modo Transparente

En este modo, el cual viene configurado por defecto, todos los datos que se reciben por la UART son enviados por la radio y análogamente lo que se recibe por la radio es enviado a través de la UART. En la Tabla 4.1 se observa la configuración necesaria para funcionar en este modo.

	Módulo 1	Módulo 2	Módulo 3	Descripción
MY	1111	2222	3333	Dirección corta del módulo en la red
DH	0000	0000	0000	Dirección larga de destino (64 bits address)
DL	2222	1111	1111	Dirección corta de destino (16 bits address)
ID	0010	0010	0010	Identificador de la red
CH	0011	0001	0001	Canal elegido para la comunicación
A1	0	0	0	Asociación de dispositivo anulada
CE	0	0	0	Coordinador deshabilitado
AP	0	0	0	Modo API deshabilitado.
RO	3	3	3	Tiempo sin recibir datos antes de enviar

Tabla 4.1: Configuración XBee en modo transparente

En el ejemplo configurado en la tabla anterior, la información que se recibe por el puerto serie de los módulos 2 y 3 se transmite hacia el módulo 1. Además, la información que se recibe por el puerto serie del módulo 1 se envía solamente al Módulo 2. Para que todos los módulos recibieran toda la información que recogen todos, hay que configurar su dirección de envío como *broadcast*: (DH = 0000, DL = 0xFFFF). Si el parámetro de dirección corta del XBee, MY es igual a FFFF este módulo será ignorado y no recibirá ningún paquete direccionado con dirección corta.

Para comprobar su funcionamiento se abre una sesión del *HyperTerminal* de Windows configurado con el puerto serie del XBee y se coloca otro módulo alejado de este con el conector de *loopback* conectado en su puerto serie. Desde el PC, a través de la sesión de *HyperTerminal* se envía en caracteres ASCII la frase, "Hola Mundo", inmediatamente se recibe la misma frase, mostrándose en el *Hyperterminal*. Para comprobar que ha sido través del XBee, a través del adaptador de *loopback* se le quita este adaptador y se reenvía la frase, observando en este caso que no hay respuesta.

4.2.2 Configuración del "IO Line Passing", Cableado Virtual

El "*IO Line Passing*" consiste en crear un vínculo entre una entrada de un módulo XBee y la salida de otro módulo, situado dentro del alcance de radio del primero, de modo que las entradas recibidas por el primer módulo se reflejen en la salida del segundo.

El objetivo de esta prueba es vincular la entrada analógica de un módulo, llamado emisor, con la salida de PWM de otro módulo, llamado receptor, de tal forma que se pueda transmitir una señal captada desde un sensor hasta otro punto sin necesidad de cableado.

Como restricción se tiene que estas señales se han de mapear emparejadas por el número de entrada/salida, es decir que DIO2 del emisor se emparejaría con DIO2 del receptor, DIO3 con DIO3 y así hasta DIO7. El caso de DIO0 y DIO1 son diferentes, al poder ser entradas analógicas o digitales. Si en el emisor se configura DIO0 o DIO1 como

entrada digital, en el receptor se obtiene en DIO0 ó DIO1 la salida digital que reciben las correspondientes entradas del emisor. Si por el contrario en el emisor se configuran DIO0 o DIO1 como entrada de ADC, en el receptor entonces van emparejadas con PWM0 (P0) o PWM1 (P1), respectivamente, que representarán el nivel de señal recibida en las entradas de ADC del emisor.

Para realizar la prueba se definen tres módulos XBee, el número 1 es el módulo base actuando de receptor y dos módulos remotos actuando de emisores que representan dos sensores colocados en áreas dentro del alcance del módulo Base. El XBee Base recibe los paquetes de información con la lectura de la entrada analógica de los dos módulos a los que está enlazado inalámbricamente y la refleja a sus dos salidas de PWM. El módulo 2, en su entrada de ADC 0, que está situada en el Pin 20, recibe cambios de una señal analógica, debidamente adaptada a la tensión de entrada del XBee (0 – 3,2 V). El módulo 3 recibe una señal analógica en el rango (0 – 3,2 V) por su ADC 1, que está situado en el Pin 19. En el módulo base se tiene la señal recibida por el módulo 2 en el PWM 0, que es su pin 6, y la señal captada por el módulo 3 en el PWM 1 que está situado en el Pin 7.

En la Tabla 4.2 se muestra la configuración mínima necesaria de los tres módulos XBee para llevar a cabo la prueba.

	Módulo 1 Base	**Módulo 2 Sensor 1**	**Módulo 3 Sensor 2**	**Descripción**
MY	1111	2222	3333	Dirección corta del módulo en la red
DH	0000	0000	0000	Dirección larga de destino (64 bits add)
DL	2222	1111	1111	Dirección corta de destino (16 bits add)
ID	0010	0010	0010	Identificador de la red
D0	0	2	0	Indica la entrada D0 como ADC
D1	0	0	2	Indica la entrada D1 como ADC
P0	2	0	0	Configuración PWM 0 – Actualiza de 2
P1	2	0	0	Configuración PWM 1 – Actualiza de 3
IR	--	0x0F	0x14	Intervalo entre muestras. 1,5 s y 2 s
IT	1	2	2	Nº Muestras antes de enviar.
IU	1	0	0	Envía la información recibida a la UART
IA	0xFFFF	1111	1111	Modulo 1 recibe cambios del 2 y 3
PT	0x50	--	--	Tiempo de remanencia de datos PWM.

Tabla 4.2: Configuración red con XBee para realizar *IO Line Passing*

En esta configuración se establece el tiempo de permanencia de las salidas de los PWM de 5 s (0x32(hex) x 100 ms), mediante el comando PT, parámetro que es común para todas la salidas, expresado en múltiplos de 100ms, para que permanezca activa hasta la siguiente muestra. Esto es así para que al tener configurado un intervalo entre muestras de 2 s en el módulo con mayor latencia y configurando que se envían los datos cada dos

muestreos, se tienen datos cada 4s, al poner 5s como tiempo de permanencia de los PWM se evita que este fije su valor a 0 entre las muestras, por sobrepasar el tiempo de permanencia de las salidas.

En la Ilustración 4.1 se observa el esquemático con las conexiones necesarias para una vez configurados los módulos XBee utilizarlos para realizar el *IO Line Passing*, donde dos potenciómetros simulan la entrada de los sensores analógicos.

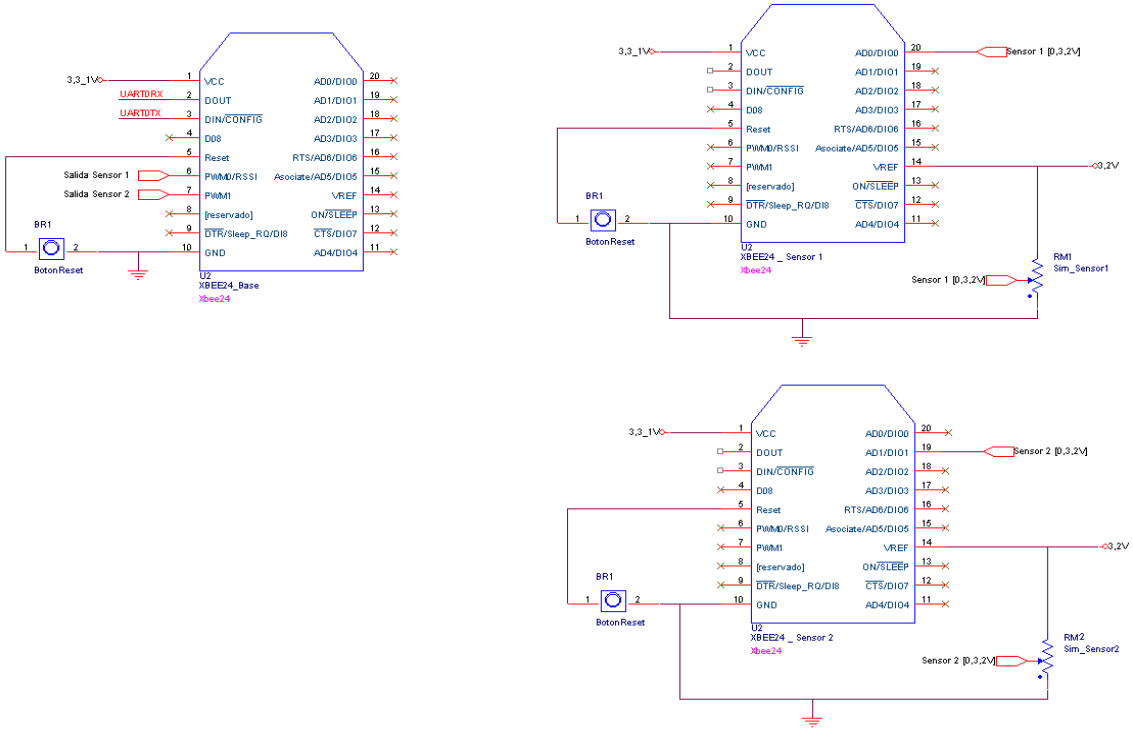


Ilustración 4.1: Diagrama de las conexiones mínimas para realizar *IO Line Passing*

4.2.3 Establecer una red con Coordinador y Detección Automática de Parámetros de Conexión, por los Dispositivos Finales.

Los módulos XBee pueden crear redes con y sin coordinador, permitiendo unirse a redes ya existentes aún cuando no se conozcan todos los parámetros de conexión de la red. En el caso práctico que se desarrolla a continuación, se crea una red con coordinador y dos XBee como elementos finales, que se asocian a la red creada por el coordinador sin conocer alguno de los parámetros de conexión de esta red.

En esta prueba se configura un XBee para que actúe de coordinador, y se disponen dos módulos actuando de dispositivos finales, situados dentro del alcance de la radio del coordinador. La ventaja de este tipo de red es que se pueden configurar periodos de inactividad en los dispositivos finales y de este modo se ahorra energía. Además, con esta

configuración de red se obtiene una mejor organización de los módulos, al poder agruparlos en diferentes redes (PAN) con cometidos diferentes.

Antes de comenzar se ha de comprobar que todos los XBee con los que se establezca una relación de maestro/esclavo mediante la asociación tienen la misma versión de Firmware, esto es importante para evitar posibles incompatibilidades entre versiones.

En la Tabla 4.3 se detalla la configuración necesaria en los XBee para crear esta red.

	Módulo 1 Coordinador	**Módulo 2 Disp. Final 1**	**Módulo 3 Disp. Final 2**	**Descripción**
MY	1111	2222	3333	Dirección corta del módulo en la red
SD	0004	0004	0004	Tiempo de escucha, para detectar el canal
CE	1	0	0	Coordinador o dispositivo final
A2	111b	000b	000b	Configuración coordinador
A1	0000b	0111b	0111b	Configuración dispositivo final
SP	0	0	0	Periodo de ciclo de Sleep. 0 No entra en modo Sleep
ST	0	0	0	Tiempo antes de ponerse en modo Sleep

Tabla 4.3: Configuración red de XBee con Coordinador y dos dispositivos finales

Configuración del coordinador

Se define en el Coordinador el PAN ID, para esta prueba se deja sin definir el canal, para que el módulo escoja automáticamente el de menor energía. Ahora se explica el significado de cada parámetro y sus posibles valores.

Parámetro A2. (Configuración del coordinador – consta de 3 bits)

- Bit 0 = 1. (*AutoPAN_Id*) Sirve para activar un escaneo de las PAN libres. Se auto-asigna un PAN Id libre. (ID)
- Bit 1 = 1. (*AutoCH_Id*) Busca el canal en el que se consumiría menor energía, este canal es el parametrizado al coordinador. Se asigna el canal (CH-*Channel)* de menor energía.
- Bit 2 = 1. Permite la asociación.

Para los propósitos de la prueba se configura del siguiente modo: PANID=110, buscara el canal de menor energía y se permite la asociación.

El parámetro CE, es el que marca el modo de coordinador o Dispositivo final.

- CE = 1 → Coordinador
- CE= 0 → Dispositivo final. (Viene por defecto)

Se configura CE=1; Para el coordinador.

Los parámetros SP (Periodo de ciclo de *Sleep*) y ST (Tiempo antes de ponerse en modo *Sleep*), deben de coincidir en dispositivos finales y coordinador para que no se pierdan datos en la comunicación.

Configuración del dispositivo final

Parámetro A1. (Configuración del dispositivo final – consta de 4 bits)

Bit 3 = 1. Consulta al coordinador cuando retorna del modo *Sleep* si tiene mensajes pendientes a través del pin 9 (*Poll coordinator on pin wake*). Esto es práctico cuando se usa comunicación indirecta.

- Bit 2= 1=> Auto Asociamiento. El dispositivo intenta asociarse con un coordinador automáticamente al iniciarse.

  0=> No se asocia, actúa con su configuración individual.

- Bit 1: 0 => Se asociará solo con coordinador que tengan el mismo PAN ID que él.

  1=> Intentará asociarte con los coordinador sin tener el cuente su PAN ID.

- Bit 0: 0 => Se asociará con un coordinador que emita en el mismo canal que el configurado.

  1 => Se asociará con un coordinador sin tener en cuenta el canal en el que este emitiendo.

El dispositivo final detecta los coordinadores activos en su área e intenta conectarse de acuerdo a su configuración, que tiene mediante los bit 0 y 1 del parámetro A1, anteriormente descritos.

En este caso la configuración de los bits de A1 queda: 1110, es decir se le indica que consulte al coordinador al salir del modo *Sleep* si tiene mensajes pendientes para él, que comience automáticamente la asociación, se le fija en el parámetro PANID a que red debe conectarse, y se deja a 0 el canal para que busque el canal del coordinador, que no se sabe cuál es pues se ha dejado que el coordinador elija el canal a usar, al iniciarse según la energía que detecte en cada uno de ellos.

Una vez que el dispositivo final ha terminado su asociación se visualiza, fijándose si el LED 6 de la placa de desarrollo esta encendido fijo, esto significa que no hay asociación, si por el contrario, el LED está parpadeando a un ritmo de 2 veces por segundo en el dispositivo final, esto significa que ya está asociado. En el coordinador el parpadeo se produce 5 veces por segundo si tiene módulos asociados. El LED rojo, de los tres LEDs

que se encuentran en el frontal de la placa de desarrollo XBID-R/U-DEV, indica que el módulo esta operativo y asociado, es decir cuando entra en modo *sleep* este se apaga.

Aparte de observar la intermitencia de los LEDs, se pueden leer los parámetros de cada módulo y observar como todos ellos tienen asignado el mismo canal, dirección de destino (DL, DH) y el PAN ID que ya se había prefijado, este no debe de haber cambiado, pues es el parámetro con que se diferencia cada subred.

En este ejemplo el coordinador y el dispositivo final están siempre activos, es decir no entran en el modo de bajo consumo (modo *Sleep*). Este funcionamiento se trata en ejemplos posteriores.

4.2.4 Modo API - Comandos Básicos

El modo API (*Application program interface*), permite leer y escribir las configuraciones de los XBee, así como tener acceso a las tramas de datos de entradas/salidas. Esta comunicación se lleva a cabo a través de su UART conectada localmente a un PC o microcontrolador, ó inalámbricamente a través de otro módulo. Para ello se han de seguir las estructuras que este API especifica.

En esta prueba se trabaja sobre un módulo XBee conectado localmente al puerto serie de un PC, al que se le escribirán y leerán parámetros de su configuración usando el API.

Primero se configura el puerto serie para que sus parámetros coincidan con los del módulo. Se elige el puerto serie donde se tiene conectado el XBee, ya sea un puerto físico o a través de USB. La configuración por defecto de los parámetros de comunicación en los módulos XBee es la mostrada en la Tabla 4.4 y es la que tienen al conectarlos por primera vez.

Velocidad (baudios):	9600
Control de Flujo:	Ninguno
Bits de Datos:	8
Paridad:	Ninguna
Bits de parada:	1

Tabla 4.4 : Parámetros de comunicación por defecto en módulos XBee

En el PC se utiliza el terminal que se encuentra en el programa "X-CTU". Este terminal es muy sencillo de usar y muy conveniente porque permite ver los bytes que se envían o recibe, en ASCII y en hexadecimal, siendo muy útil cuando se trabaja en el modo API, para poder identificar las tramas.

Una vez que se tiene en el X-CTU abierta la ventana de terminal, automáticamente abre el puerto con los datos que se especifican en la primera pantalla. Si se ha abierto el puerto correctamente la ventana aparece con el fondo en blanco (en la Ilustración 4.2 se muestra el resultado), si se cierra el puerto o no ha podido abrirse el fondo de la pantalla aparecerá en gris y no deja realizar modificaciones en ella. Esta pantalla es similar a la del *HyperTerminal* de Windows, pero con la salvedad que se puede elegir tener un divisor que permite tener, a un lado los caracteres en ASCII y al otro lado los dígitos hexadecimales que se usaran en el modo API.

Para entrar en modo comando del XBee se ha de teclear en la parte ASCII la secuencia de entrada, que por defecto es "+++", se deja pasar un tiempo de guardia (2s) sin introducir nada más y el módulo responde con el mensaje de "OK" cuando haya entrado en el modo comando o nada si no ha podido entrar con lo que se debe de esperar unos segundos (2s) y volver a intentarlo.

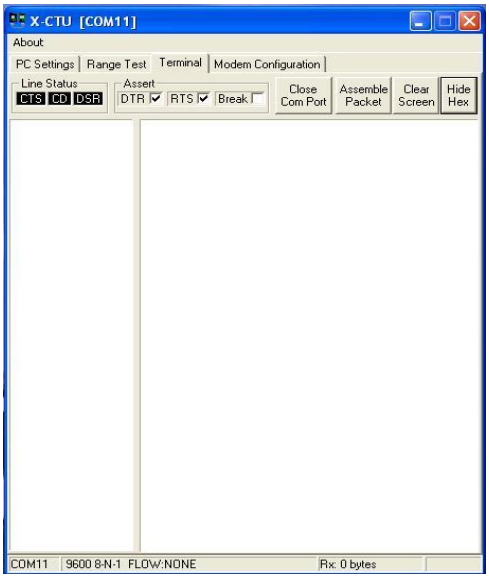


Ilustración 4.2: Pantalla de terminal X-CTU

Una vez que el módulo está en el modo comando ya se le pueden enviar comandos AT con el formato visto en la sección de comandos AT. Se recuerda que los comandos pueden leerse si se envían sin parámetros y escribirse si se envían seguidos de un valor.

A continuación en el Ejemplo1, se describe como se consulta el canal de transmisión del XBee, se le modifica y se comprueba que la modificación ha tenido lugar volviendo a consultar el canal de transmisión del módulo.

Ejemplo 1:

→ +++	// Secuencia de entrada
← OK	
→ ATCH	// Se envía el comando de consulta del canal, como no lleva parámetros solo se lee
← C	// El módulo responde que tiene configurado el canal 0xC.
→ ATCH=D	// Se cambia el canal a 0xD
← OK	// Como el cambio lo realiza correctamente responde con OK
→ ATCH	// Se vuelve a consultar el canal
← D	// Como respuesta se obtiene que está configurado el canal 0xD.
→ WR	// Se debe de enviar el comando WR para que los cambios queden
	// reflejados en la memoria no volátil y no se pierdan al resetear.
← OK	// Si ha guardado bien los parámetros lo indica el con el mensaje "OK".

En la Ilustración 4.3 se muestra el ejemplo que se describe anteriormente, realizado sobre un XBee conectado a un PC por el puerto serie, donde se aprecian los envíos hacia el módulo XBee en azul y las respuestas de este en color rojo.

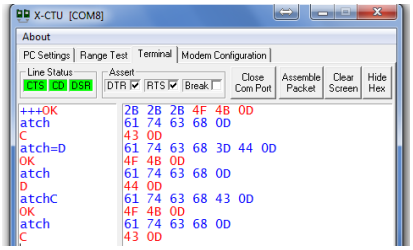


Ilustración 4.3: Comandos AT en Terminal X-CTU

Una vez que el módulo esta en modo comando AT, para configurar el módulo en modo API, se envía el comando AP=1.

–> +++	// Secuencia de entrada modo AT
← OK	
→ AP = 1	// Pasa el XBee a modo API.
← OK	
→ ATCN	// Guarda los cambios

Tras esta secuencia ya está el XBee en modo API y cualquier byte o carácter que se envíe al módulo por el puerto serie lo interpretará como una trama API.

Se crea una trama API de consulta del estado del módulo. Ayudándose de la opción de enviar un paquete completo del software X-CTU, (Se encuentra activando el botón *Assemble packet,* crear paquete) se envía toda la trama seguida, además se le indica que se va a escribir directamente en hexadecimal los valores y no en ASCII.

Se crea una trama de la UART que contendrá el comando API, siguiendo la estructura de la Ilustración 4.4, vista en el capítulo 3.

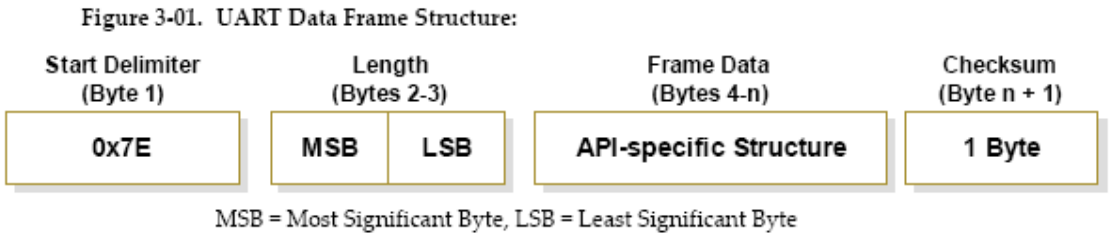


Ilustración 4.4: Estructura de las tramas UART

La trama de la UART a enviar, en hexadecimal queda: 7EXXYYAPICC

El significado de los bytes que la componen es:

0x7E	Es el delimitador de campo UART.
XX	Es el byte de mayor peso de la longitud del campo de datos de la trama.

YY	Es el byte de menor peso de la longitud del campo de datos de la trama.
API	Trama API a enviar, se define en el siguiente paso.
CC	1 Byte de checksum.

Ahora se genera la trama API necesaria para la consulta del canal de transmisión que tiene configurado el módulo sobre el que se trabaja. Se genera una trama conteniendo el comando ATCH, para la consulta de canal CH, *Channel*.

Se utilizará la estructura de una trama API de envío de comando AT, mostrada en Ilustración 4.5.

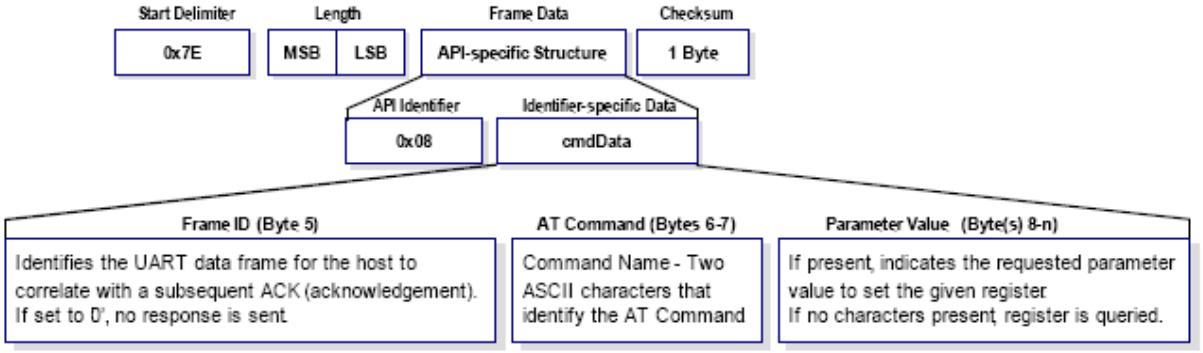


Ilustración 4.5: Estructura API de envío de comandos AT

En hexadecimal se tienen los siguientes dígitos: 0x08 0x01 0x43 0x48 cuyo significado se detalla:

0x08	: Identificador de tipo de trama API, trama de envío de comando AT.
0x01	: Identificador de trama, si es 0 no se enviará una respuesta.
0x43	: El carácter C en hexadecimal.
0x48	: El carácter H en hexadecimal.

No se indican parámetros pues solo se quiere realizar una consulta.

Se procede a calcular el resto de campos de la trama UART. La longitud del campo de datos de la trama UART, que se es el valor de la longitud de la trama API completa, en este caso se compone de 4 bytes, por tanto los campos MSB y LSB quedan como:

MSB:	0x 00	// Mayor peso del tamaño del *Payload* de la trama UART
LSB:	0x04	// Menor peso del tamaño del *Payload* de la trama UART
API:	0x08 0x 01 0x43 0x48	// Trama API de envío del comando CH.
CC:	0x6B	// Checksum

Para el *checksum* se suman los bytes del campo de datos (*payload*) de la trama UART, se usa el byte de menor peso de la suma y se resta a 0xFF, lo obtenido es el byte de *checksum*. En el ejemplo actual se calcula en la siguiente expresión:

0x FF – (0x08+0x01+0x43+0x48) == 0xFF - (0x94) = 0x6B

Por tanto la trama completa a enviar en hexadecimal queda:

7E 00 04 08 01 43 48 6B

En la Ilustración 4.6, en color rojo se observa la respuesta por parte del XBee, esta trama de respuesta es un paquete UART que en su campo de datos contiene una trama API de respuesta de comando AT, en esta trama devuelve el canal que el módulo tiene configurado, en este caso el 0x0C como se desglosa en la explicación de la trama.

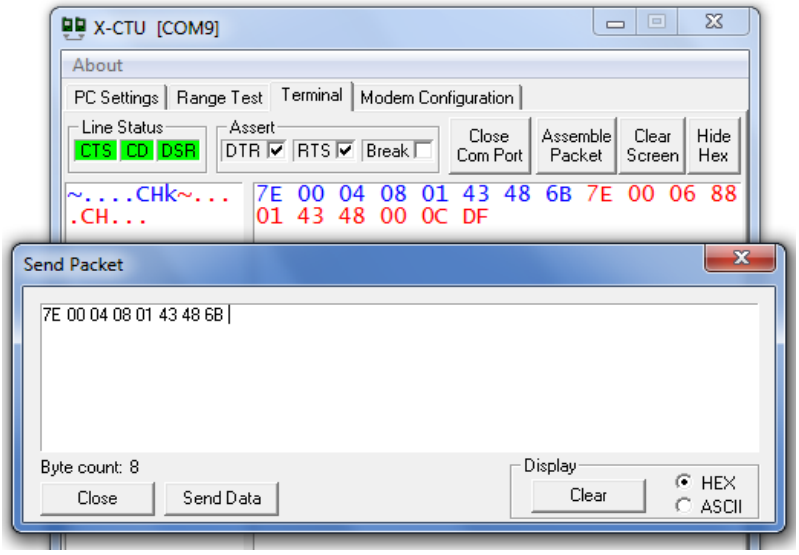


Ilustración 4.6: Terminal X-CTU con la trama enviada (azul) y la recibida (rojo)

La trama recibida es: 7E 00 06 88 01 43 48 00 0C DF.

7E	: Delimitador de cabecera de trama UART.
00 06	: MSB y LSB. La trama de datos del paquete UART consta de 6 bytes.
88	: Tipo de trama API, 0x88 Trama de respuesta de comando AT.
01	: Identificador de la trama como se observa responde con el mismo : identificador de trama que se le envió.
43 48	: Comando AT al que responde en hexadecimal: 43 (C), 48 (H), CH
00	: Estado de la respuesta: 0 indica que todo está OK.
0C	: Indica que tiene configurado el canal 0xC.
DF	: Byte de *checksum*, se comprueba, 0x88+0x01+0x43+0x48+00+0C+DF = 1FF : la suma de los bytes da 1FF, si solo se tiene en cuenta el menor byte : se tiene 0xFF indicando que la trama esta correcta.

Ya se ha realizado el primer envío de un comando mediante una trama API y también mediante el modo comando. Se observa que para hacer cambios en el módulo conectado es mucho más simple hacerlo mediante la introducción directa del comando AT. No obstante el uso del API será imprescindible para la realización de las siguientes pruebas, ya que con el modo API se podrán enviar comandos AT al mote que tenemos conectado y también a motes en el alcance de la radio de este, mediante el uso de la trama de API de comando remoto. También para configurar las entradas/salidas, donde es imprescindible usar el modo API.

Viendo la importancia que tiene trabajar en los XBee mediante el API y teniendo en cuenta que el trabajo de hacerlo a mano es muy tedioso y lento, al tener que trabajar siempre en hexadecimal, calcular longitudes, calcular el *checksum*, tanto en el envío como para interpretar las respuestas por parte del XBee. Se procede al desarrollo de la primera versión de una aplicación en Java que permite generar estas tramas de los comandos de forma automática, así como decodificar las respuestas por parte del XBee. En la Ilustración 4.7 se observa una captura de pantalla de esta aplicación.

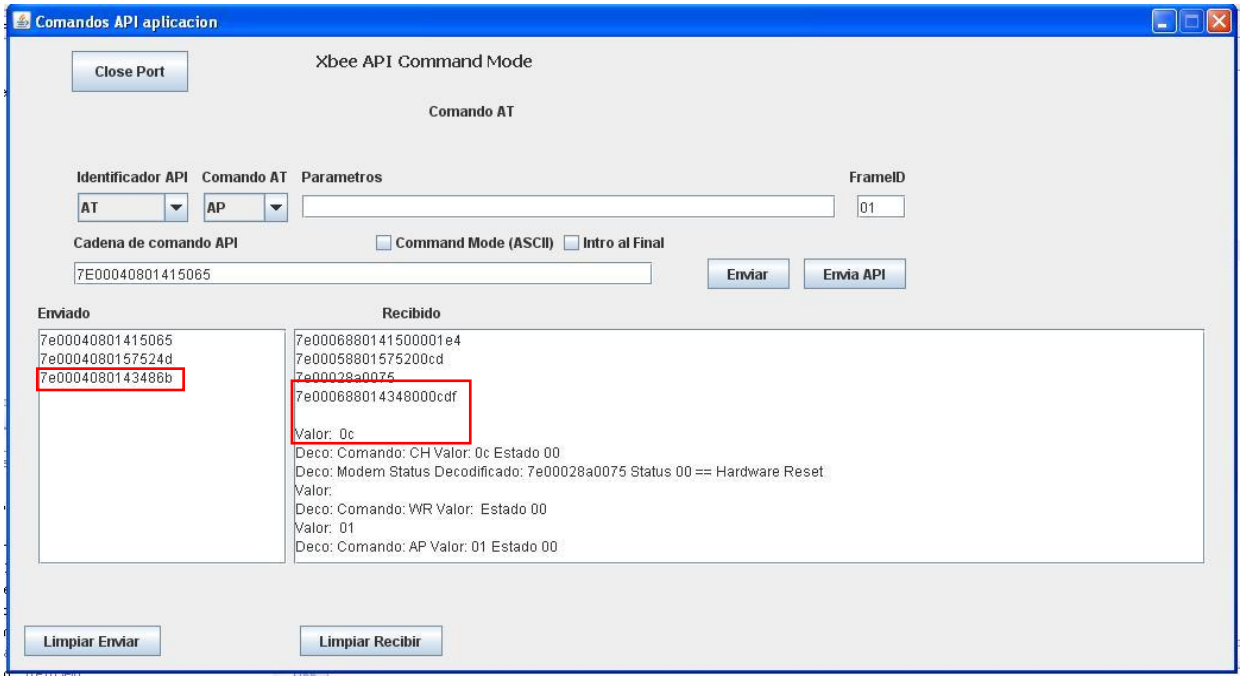


Ilustración 4.7: Screenshot de la primera aplicación Java de generación de tramas API

Como se observa en la Ilustración 4.7, la aplicación consta de un cabecera donde se pueden enviar comandos al XBee por el puerto serie y en la parte inferior consta de dos ventanas. La ventana de la izquierda donde aparece todo lo que se ha enviado al XBee y la de la derecha donde van apareciendo las respuestas recibidas desde el XBee con su correspondiente decodificación.

La primera versión del programa se desarrolló de tal manera que fuera capaz de enviar cualquier comando AT mediante el modo comando del XBee y también fuese capaz

de enviar tramas API con comandos AT y sus parámetros, además de decodificar la respuesta del XBee.

En una segunda fase se le ha añadido la opción de decodificación de las entradas digitales y analógicas captadas por el XBee.

Funcionamiento de la aplicación para envíos de comandos AT

En la aplicación para enviar un comando AT, se debe de marcar la casilla de *Command Mode* (ASCII), también la de *Intro* al final y escribir los caracteres en el campo de texto "Cadena de comandos API" para entrar al modo Comando del XBee, por defecto se usa "+++", el *Intro* ya lo introduce la aplicación una vez que se pulsa el botón enviar. Una vez recibida la confirmación de "OK" por parte del XBee ya se puede enviar y recibir comandos como se vio en los ejemplos anteriores cuando se usaba el terminal serie, con la peculiaridad de dar al botón enviar una vez se tenga formado el comando.

Funcionamiento para envíos de tramas API

La parte más interesante de este primer desarrollo de programa de comunicación con el XBee viene de la posibilidad de elegir en los menús desplegables de la aplicación el tipo de comando API y utilizar los campos de texto para completar el comando y la aplicación generará la cadena hexadecimal para enviar al XBee. Una vez enviada al XBee se recibe la respuesta que será decodificada por la aplicación. De esta manera se puede empezar a testear el funcionamiento de los comandos API sin tener que decodificar las respuestas, cada vez que se reciben o envían.

En la Ilustración 4.7 mostrada anteriormente de la aplicación se observan los parámetros seleccionados para enviar el comando CH de tipo AT, para consultar que canal de comunicación está configurado el módulo.

Para este comando el programa ha formado la siguiente trama:

0x 7E0004080143486b

Una vez ejecutada por parte del XBee se recibe la siguiente trama:

0x7E000688014348000cf

La decodificación y significado de la respuesta es la siguiente:

0x7E	Delimitador.
0x00006	Tamaño 6 bytes.
0x88	Tipo de comando. (0x88 = Respuesta de comando AT).
0x01	Identificador de la trama 1.

0x43 0x48	Comando al que responde CH.
0x00	Estado de la respuesta => OK.
0x0C	Respuesta (canal 0xC), *Checksum*: 0x0f.

La Ilustración 4.8, muestra una captura de pantalla de la aplicación donde se puede observar como al elegir otro tipo de comando diferente a enviar cambian los campos que se han de rellenar adaptándose al comando a enviar. En este caso se ha elegido enviar un paquete de datos con dirección corta. Para ello se han habilitado en la aplicación los campos de dirección, parámetros, que este campo es donde se introducirán los datos a enviar con una longitud máxima de 100 caracteres y el campo *Frame ID* que es el número que identifica a la trama.

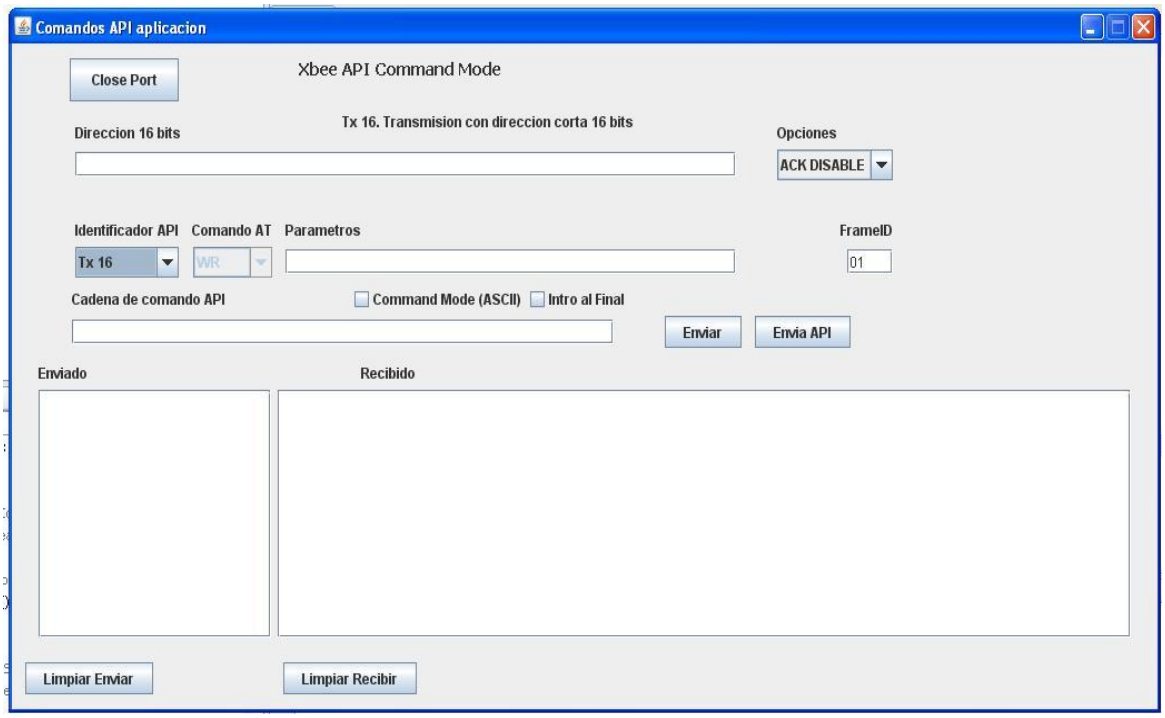


Ilustración 4.8: Aplicación para la generación de tramas de envío de datos

La aplicación se ha desarrollado en Java por ser un lenguaje multiplataforma, ampliamente utilizado, orientado a objetos y con grandes capacidades para la comunicación. Aparte de esto, es un producto de libre distribución lo que hace que disminuyan los costes de desarrollo. Además, posee la ventaja de poder contar con IDE (*Integrate Development Environment*- Entornos de desarrollo) como Eclipse o Netbeans de alta calidad que se pueden conseguir bajo licencias de libre distribución. En este proyecto se han usado ambos IDEs, Netbeans en el desarrollo inicial, y en una segunda fase se migro el desarrollo a Eclipse para el desarrollo de las interfaces de usuario.

Para el desarrollo eficiente de la aplicación se ha creado una librería con los distintos tipos de comandos API que están definidos en el manual de usuario del XBee [1], tanto de envío como de respuestas, para así poder enviarlos y decodificarlos

automáticamente como tramas API. En el desarrollo de la aplicación se han usado las propiedades de encapsulación de Java para crear un tipo Comando y luego que este pudiera acoger a cualquiera de los subtipos de comandos API que se reciben, y codificarlos o decodificarlos cada uno según marcan sus estructuras.

En el capítulo 5, en el apartado de creación de la librería API, se explica el desarrollo de la librería de la aplicación y se detallan sus clases y las relaciones entre ellas. Esto se detalla en el siguiente capítulo porque la librería es la misma para la primera aplicación como para la aplicación del caso de estudio y se ha ideado de manera que pueda ser reutilizada en futuras aplicaciones con módulos XBee.

4.2.5 Prueba del API para Entradas

Un uso importante del API es para utilizar las entradas de ADCs (*Analog to Digital Converter*, Conversión analógica digital) que incorpora el XBee, y las entradas digitales ya que estas siempre se devuelven en una trama API.

Hay dos tipos de tramas que pueden ser recibidas, una trama para dirección de 16 bits y otra para trama para dirección de 64 bits. Los identificadores de la trama API son, 0x82 para tramas con dirección de 16 y 0x83 para dirección de 64 bits.

Las tramas de entrada/salida tienen la misma cabecera que una trama de respuesta de datos de 16 y 64 bits, respectivamente. Por ejemplo, para una trama de 16 bits, la cabecera sería la mostrada en la Ilustración 4.9, con la diferencia que Identificador API es 0x82 ó 0x83 dependiendo del tipo de direccionamiento.

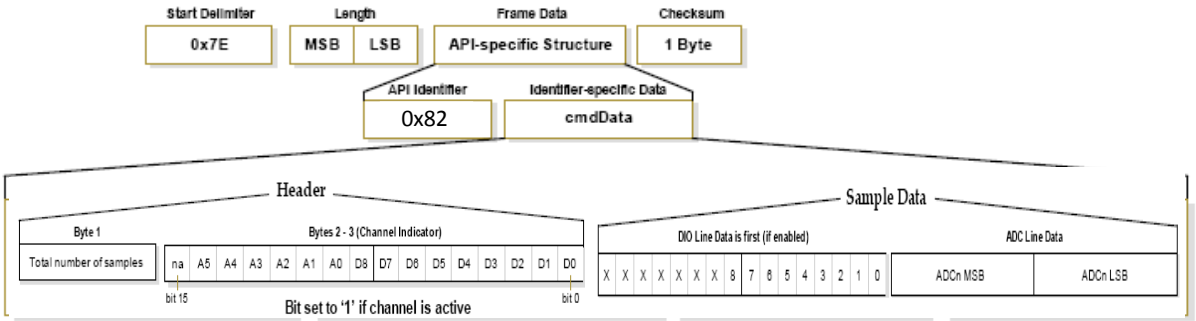


Ilustración 4.9: Trama contenedora de los datos de entradas/salidas

En el campo de datos (cmdData) está contenida la información de las entradas, dividido en dos subcampos, el primero es la cabecera, mostrada en la Ilustración 4.10, donde está una máscara indicando que entradas están activas y el número de muestras que contiene el paquete. El segundo subcampo mostrado en la Ilustración 4.11, es donde aparece la información del estado de dichas entradas, primero de las entradas digitales si las hubiera y después de las analógicas.

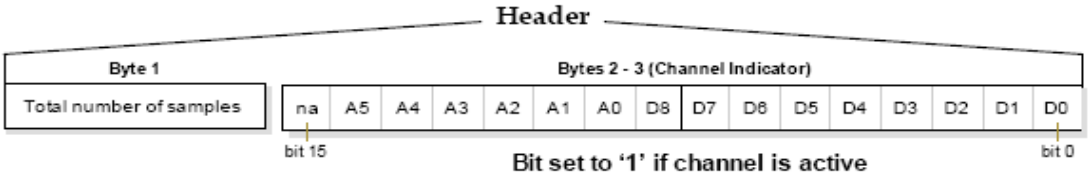


Ilustración 4.10: Formato de la cabecera de datos de Entrada

La estructura del "*Sample Data*", Ilustración 4.11, se repite para cada muestra que se haya enviado en el paquete, es decir, si se han recogido cuatro muestras con información digital y analógica, esta estructura aparece cuatro veces, una muestra seguida de la siguiente, es decir que la cabecera I/O (*Input/Output*- Entradas/Salidas), Ilustración 4.10, solo aparece la primera vez para todas las muestras y luego se repite el campo de datos de I/O.

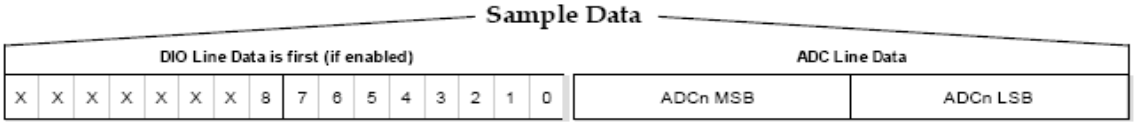


Ilustración 4.11: Formato del cuerpo de los datos de Entrada

En la Tabla 4.5 se indica la configuración mínima para tener un módulo XBee recogiendo datos automáticamente con un periodo de 5s, y otro módulo conectado al PC mediante puerto serie recibiendo dichos datos y mostrándolos por pantalla.

	Módulo 1 Coordinador	**Módulo 2 Disp. Final 1**	**Descripción**
MY	1111	5001	Dirección corta del módulo en la red
ID	3332	3332	PAN ID. Igual en el Coordinador y el EndDevice
SM	0	0	Módulos no entran en modo Sleep
CE	1	0	Coordinador habilitado.
A1	0	0110b	En Disp. Final se configura para asociarse.
A2	011b	0	El Coord. Se configura para asociar a módulos con su mismo PanId.
AP	1	1	Habilitación modo API.
IR	0	1338	Periodo de muestreo 5 s (máx. 65 s)
IT	1	2	Nº Muestras antes de enviar
ATDn	0	ATD2=2 ADT3=3	0 Desactivada **2 Entrada Analógicas** **3 Entrada Digital** 4 Salida digital Low 5 Salida digital High

Tabla 4.5: Configuración red con XBee para lectura de IO con API

En el ejemplo siguiente con la configuración de los XBee descrita en la tabla 4.5, se lee una entrada digital en el pin 18 y otra analógica en el pin 20, cada 5 segundos, y se envía

el paquete de datos cuando tiene recogidas dos muestras, por lo que en el módulo base se recibe un paquete con dos muestras cada 10s.

Primera trama recibida tras la prueba:

7E 00 10 83 50 01 3A 00 02 08 08 00 08 02 54 00 08 02 54 23

El significado de los bytes de la trama recibida es el siguiente:

0x 7E : Cabecera
0x 00 10 : Tamaño de la trama 16 bytes
0x 83 : Identificador de recepción de IO con dirección de 16 bits
0x 50 01 : Dirección de origen 5001
0x 3A : RSSI
0x 00 : Opciones 0=OK
// Comienza trama de IO
0x 02 : Número de muestras (2 muestras)
0x 08 08 : Mascara de canales activos: 0000 1000 0001 0000: Canales activos=> A2 y D3
0x 00 08 : Estado de las entradas digitales, D3 en estado activo.
0x 0254 : Valor de la entrada analógica nº 2. (254 / 3FF) * 3,3 V = 1,9 V } 1º Muestra
0x 00 08 : Estado de las entradas digitales, D3 en estado activo.
0x 0254 : Valor de la entrada analógica nº 2. (254 / 3FF) * 3,3 V = 1,9 V } 2º Muestra
// Fin de datos IO
0x 23 : Valor del *Checksum*

De la trama obtenida tras la ejecución se observa que en la entrada analógica se tiene un valor de 254 en las dos muestras que contiene la trama, el valor máximo de las entradas analógicas es 0x3FF (10 bits de resolución), que equivale a la tensión de referencia. En este caso la tensión de referencia es igual a la de alimentación, que está fijada a 3,3 V. Por tanto, si se realiza la escala (254/ 3FF), pasado a decimal se obtiene el cociente 0,583 * 3,3 = 1,92 V en la entrada analógica. La entrada digital no está activada y aparece a 1 porque no tiene tensión y tiene activas las resistencias de *pull-up*. Si se pulsa el interruptor se redirigía a tierra y se obtendría un valor de 0 en las tramas.

En la Ilustración 4.12 se muestra un ejemplo donde se tienen dos tramas recibidas con la información de dos entradas digitales y una analógica, leídas en diferentes momentos. El primer paquete recibido esta rodeado por el cuadro color azul, y el segundo paquete aparece rodeado por un cuadro color fucsia (ambas tramas contienen dos muestras).

Descartando los 8 primeros bytes, que son los propios de la cabecera de una trama de recepción de I/O, se tiene en primer lugar y subrayado en verde, el byte con el número de muestras enviadas en el mismo paquete (0x2), 2 en este caso. En los 2 bytes siguientes, marcados en amarillo, se observa como la cabecera de la trama de entradas/salidas es la misma en los dos paquetes. En ambos casos es 0x08 18, pues en las dos se monitorean las entras digitales D3 y D4 y la entrada analógica A2.

En los bytes correspondiente al valor de las entradas digitales (byte 12 y 13 para la primera muestra y bytes 17, 18 para la segunda) se tiene un valor de 0x 00 18 en ambas muestras de la trama 1. Si se ejecuta la operación de Y lógico en binario con la máscara de entradas activas (0x0818) y el valor recibido (0x0018), se obtiene, el valor 0x0018, representado en binario es 0b11000, significando que D3=1 y D4=1, no se han activado ninguna de ellas.

En la segunda trama se tiene el valor de entradas digitales 0x 00 08 (Din 3, 4 s1 y Din 3, 4 s2 en la Ilustración 4.12). También en las dos muestras, si lo comparamos con la máscara de entradas/salidas digitales activas (0x 08 18), tenemos que D3=1, es decir no está activa y D4=0 si esta activa, siendo la diferencia entre ambas tramas de haber activado la entrada digital D4. La entrada analógica (Ain 2 s1 y Ain 2 S2 en la Ilustración 4.12) no ha variado ni entre muestras ni entre tramas teniendo el valor en este ejemplo de 0x339 (80% de Vref).

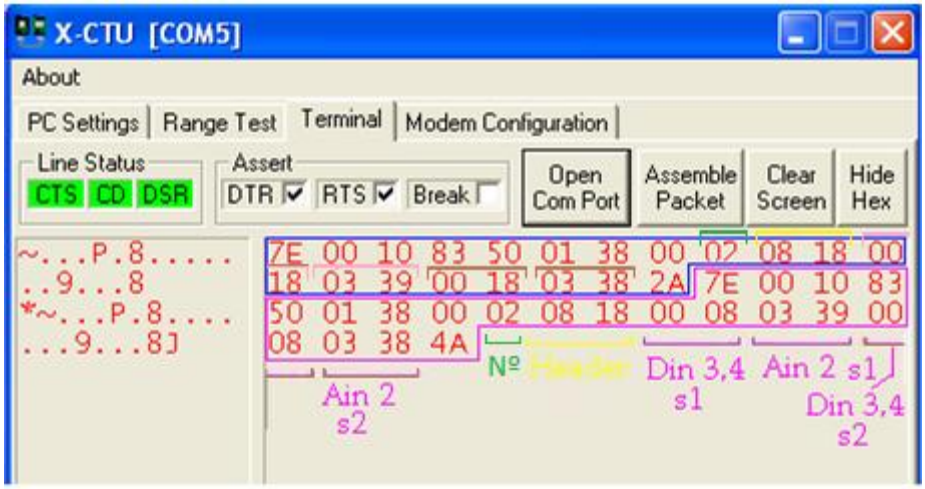


Ilustración 4.12: Desglose de tramas de Entrada / Salida recibidos por la UART

En la Ilustración 4.13 se observa el mismo resultado visto en la Ilustración 4.12, pero obtenido con el programa de decodificación realizado en java para estas mismas entradas. Se muestra que es mucho más cómodo ver el valor de las entradas al no tener que aplicar las máscaras y presentar la información ya decodificada, indicando el valor y si pertenecen a un paquete con varias muestras, el número de la muestra.

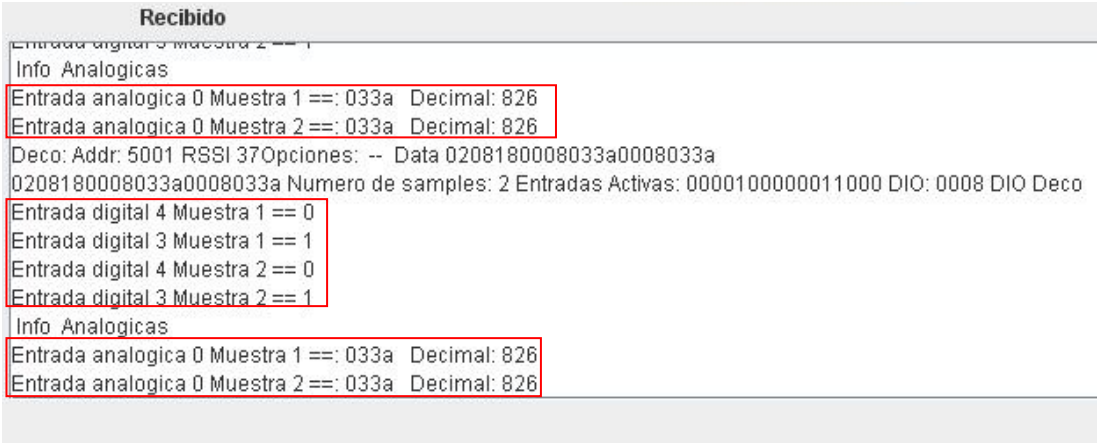


Ilustración 4.13: Entradas API decodificadas mediante la aplicación

4.2.6 Entradas con Modo Sleep

Debido a que el gran uso de estos módulos se realiza utilizando el API para realizar tareas de monitorización de sensores alejados o que cambian de ubicación con frecuencia, es importante que la autonomía de ellos sea lo mayor posible. En la prueba anterior se vio como es posible obtener los datos desde los sensores, tanto analógicos como digitales, pero tiene el gran problema de estar el módulo siempre activo para poder enviar y recibir, con lo que su consumo resulta elevado si se quiere que se mantenga alimentado con una batería durante años.

Los datos dados por el fabricante de consumos en los diferentes estados son:

- Transmitiendo: 45 mA
- Recibiendo / sin actividad : 50 mA
- Suspendido: < 10 uA

Se ha comprobado que en estado de recepción con la placa de desarrollo y los LEDs de esta encendidos, consume 86 mA si está recibiendo datos, y 72 mA cuando esta activo pero no está recibiendo. Cuando entra a modo suspendido o *Sleep,* el consumo del XBee con su placa de desarrollo es de 0,33 mA. Estos consumos medidos son superiores a los dados por el fabricante, pero se ha de tener en cuenta que también está incluido en la medición el consumo de la placa de desarrollo.

Dado el hecho, que no es necesario mantener el módulo activo entre diferentes muestras, se va a proceder en el siguiente ejemplo a realizar las mismas tareas que anteriormente pero teniendo al módulo en estado *Sleep* durante el tiempo entre muestras.

Si el tiempo entre muestras es mayor a 1 minuto, es un inconveniente de este módulo el no poder fijar el intervalo entre muestras a un valor mayor de 65 segundos.

Como solución para muestrear con periodos mayores de 1 minuto, se puede utilizar el modo *sleep* para tener mayores periodos entre muestras. De este modo, utilizando el modo cíclico de *sleep* se pueden tener periodos de hasta 4,5 minutos (máx. 368 segundos). Si se quieren tomar muestras en intervalos de tiempos más largos se ha de recurrir a temporizadores externos que activen al módulo exteriormente mediante el pin 9 (Sleep-rQ), que sirve para controlar los ciclos de *Sleep* del XBee, si esta activo este tipo de control mediante el parámetro "*Sleep Mode*-SM" configurado a 1.

A continuación se detallan los dos diferentes tipos de comunicación que se pueden usar, comunicación directa o indirecta:

Comunicación directa

En el primer ejemplo se usa la comunicación directa por el coordinador, es decir, que el coordinador envía los mensajes inmediatamente, sin tener en cuenta si el módulo esta en modo Sleep o no. En caso que el módulo destino este suspendido, los paquetes se perderán. Este modo se configura fijando en el coordinador el parámetro ciclo de *sleep* igual a 0 (SP=0). En la Tabla 4.6 se muestra la configuración necesaria para los XBee. En amarillo se han resaltado las diferencias con respecto al ejemplo anterior y que son las que configuran el modo *sleep* en los módulos que actúan de dispositivos finales.

	Módulo 1 Coordinador	**Módulo 2 Disp. Final 1**	**Descripción**
MY	1111	5001	Dirección corta del módulo en la red.
ID	3332	3332	PAN ID. Igual en el Coordinador y el EndDevice.
SM	4	4	Módulos en modo Sleep periódico.
SP	0	0x7D0	Periodo de Sleep. Cuando pasa a modo activo hace un barrido de IO. (20 s)
ST	0x200	0x200	Tiempo sin actividad antes de ir a sleep. (0,5s)
CE	1	0	Coordinador habilitado.
A1	0	0110b	En Disp. Final se configura para asociarse.
A2	011b	0	El Coord. Se configura para asociar a módulos con su mismo PanId.
AP	1	1	Habilitación modo API.
IR	0	1338	Periodo de muestreo 5 s. (máx. 65 s)
IT	1	2	Nº Muestras antes de enviar.
ATDn	0	ATD2=2 ADT3=3	0 Desactivada 2 Entrada Analógicas 3 Entrada Digital 4 Salida digital Low 5 Salida digital High

Tabla 4.6: Configuración red con XBee para lectura de IO con pasando a modo *Sleep*

Esta configuración de coordinador con los dispositivos finales solo sería válida para redes donde los dispositivos finales solamente envían información al coordinador pero no reciben información. Por tanto, no es recomendable trabajar con comunicación directa por

parte del coordinador si en la red hay dispositivos finales que pasan parte de su tiempo en modo Sleep, pues limita a la comunicación al sentido desde los dispositivos finales al Coordinador.

Con la configuración de la Tabla 4.6 se consigue tener un módulo coordinador con comunicación directa y un módulo dispositivo final, que pasa a modo *Sleep* después de recoger dos muestras (5 s+0,5 s), y permanece en estado suspendido 20s. Por tanto, en el coordinador recibirá un paquete con dos muestras cada 25,5 s.

Al ejecutar este ejemplo se ha obtenido el esquema de tiempos del proceso de toma de muestras y de sleep, mostrado en la Ilustración 4.14.

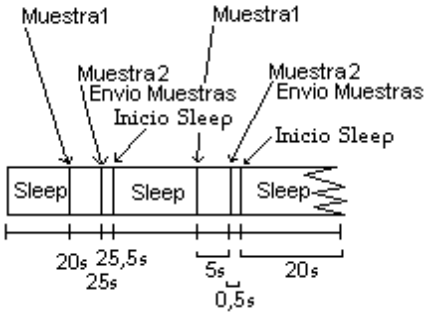


Ilustración 4.14: Representación de tiempos en los muestreos del dispositivo final

Se ha comprobado por problemas tenidos durante la realización de esta prueba, que si se tiene un tiempo muy pequeño al módulo en modo activo, es muy difícil cambiar sus parámetros, ya que al estar la mayoría del tiempo está en modo *Sleep* no se puede establecer comunicación con él, teniendo que reiniciarlo y restaurar la configuración inicial para poder cambiar sus parámetros.

Comunicación indirecta

El segundo tipo de comunicación que puede tener el Coordinador es la comunicación indirecta. Se configura mediante el parámetro SP (*Sleep Period* - Periodo de sleep) a un valor distinto de cero. Esto hace que ahora el coordinador envíe mensajes indirectamente, es decir, esperará a que los dispositivos finales le indiquen que están activos y le hagan una petición de los mensajes pendientes, entonces el coordinador se los enviará. A partir del primer mensaje la comunicación con este dispositivo final será directa hasta que este se vuelva a su estado *sleep*.

El parámetro SP ahora define el tiempo máximo, que se fija al doble del valor de SP (tmax= 2*SP), que el coordinador retiene los mensajes antes de descartarlos. Debido a esto, es importante que los parámetros ST y SP sean iguales en todos los dispositivos de la red, tanto en el coordinador como en los dispositivos finales. Aunque el coordinador use mensajes indirectos este módulo nunca entra en modo suspendido, es decir el Coordinador será el módulo de mayor consumo de toda la red.

En la Tabla 4.7 se muestra la configuración necesaria para que los módulos XBee que actúan como dispositivos finales y el coordinador, actúen en modo suspendido entre muestreos y con comunicación indirecta. Los cambios que afectan a la definición del modo *Sleep* están resaltados en color amarillo y los cambios que diferencian el modo de comunicación directa e indirecta están marcados en color azul. Esta es la principal diferencia con la prueba anterior realizada con comunicaciones directas.

	Módulo 1 Coordinador	**Módulo 2 Disp. Final 1**	**Descripción**
MY	1111	5001	Dirección corta del módulo en la red.
ID	3332	3332	PAN ID. Igual en el Coordinador y el EndDevice.
SM	4	4	Módulos en modo Sleep periódico.
SP	0x7D0	0x7D0	Periodo de Sleep. Cuando cambia a modo activo hace un barrido de IO. (20 s)
ST	0x200	0x200	Tiempo sin actividad antes de ir a sleep. (0,5s)
CE	1	0	Coordinador habilitado.
A1	0	1110b	En Disp. Final se configura para asociarse. Bit 3, indica que haga petición de mensajes al coordinador al despertar.
A2	011b	0	El Coord. Se configura para asociar a módulos con su mismo PanId.
AP	1	1	Habilitación modo API.
IR	0	1338	Periodo de muestreo 5 s. (máx. 65 s)
IT	1	2	Nº Muestras antes de enviar.
Dn	0	D2=2 D3=3	0 Desactivada 2 Entrada Analógicas 3 Entrada Digital 4 Salida digital Low 5 Salida digital High

Tabla 4.7: Configuración red con XBee para lectura de IO mediante comunicación indirecta

Al ejecutar este ejemplo se obtiene el mismo esquema de tiempos del ejemplo anterior representado en la Ilustración 4.14, obteniéndose un paquete con dos muestras cada 25,5 s, con la diferencia que si ahora el coordinador necesita enviar paquetes a los dispositivos finales los enviará, cuando pasen a modo activo, con la restricción de que como máximo puede encolar a la espera de ser enviados, dos paquetes.

4.2.7 Utilización de Salidas Digitales y Salidas por PWM

El objetivo de esta prueba es utilizar las salidas PWM (*Pulse Width Modulation*, Modulación ancho de pulso), y las salidas digitales de los módulos XBee para poder controlar dispositivos externos.

En el primer ejemplo se ejecutan los comandos necesarios para activar estas salidas en modo local, es decir se envían los comandos AT necesarios por medio del terminal serie, para comprobar su correcto funcionamiento.

Lo primero es configurar el módulo a utilizar como módulo actuador, es decir, configurar el comportamiento de sus salidas analógicas y digitales.

En el ejemplo se configura el PWM0, pin 6 del XBee como salida analógica y el D1 activo a nivel bajo, pin 19 del XBee como salida digital, ya que es posible configurar que esté en estado alto o bajo por defecto. Esta configuración sirve para definir el estado de la salida antes de recibir ninguna señal de estado y también el estado que activará cuando se pasen los temporizadores de validez de salida digital (Tn).

En comandos AT el valor del PWM0 se fija con el comando M0 (PWM0 *Output Level*, Nivel de salida de PWM0).

Para llevar a cabo esta prueba se usa la configuración vista anteriormente en la tabla 4.3, en la que se debe de cambiar los parámetros mostrados en la Tabla 4.8:

Parámetro	Valor de parámetro	Descripción
P0	2	Se configura el pin como Salida PWM.
D1	4	Se configura la DIO1 como salida digitales, con valor bajo por defecto.
D2	5	Se configura la DIO2 como salida digitales, con valor alto por defecto.
T1, T2	0x64 (10 s)	Valor de validez de la salida digital, cuando finaliza vuelve a su valor por defecto. Máximo 25,5 s
PT	0x64 (10 s)	Valor de validez de la salida PWM0, cuando finaliza vuelve a 0. Máximo 25,5 s

Tabla 4.8: Configuración red con XBee para utilización de salida digital y PWM

Ahora desde la ventana de terminal serie se van a usar los comandos M0 (*PMW0 Output Level* – Nivel salida de PWM) e IO (Digital *Output Level* - Nivel de Salida digital) para fijar el valor de las salidas digitales.

Para fijar las salidas digitales se usa el comando IO, seguido por un parámetro que es un número hexadecimal representando dos bytes, en los cuales cada bit representa el estado de las salidas digitales a activar.

En el ejemplo se usará la activación de la salida 2 y de las salidas 1 y 2, así las máscaras quedarían de la siguiente forma:

Mascara activación salida 2: 2 → 0000 0010 → en hexadecimal 0x02

Mascara activación salida 1y 2: 1,2 → 0000 0011→ en hexadecimal 0x03

Para fijar la salida de PWM se utiliza el comando M0 seguido de un número hexadecimal de dos dígitos, que indica el valor de la salida. Los valores a utilizar tienen que estar entre 0x0 (nivel 0 V) y 0x3FF (nivel máx. = Vcc), que será el valor máximo.

En el siguiente código se hace uso de ambos comandos, IO y M0.

```
→ +++                  // Se entra en modo comando
← OK                   // Confirmación de que se está en modo comando
→ ATIO0                // Se ponen las dos salidas a 0.
← OK
→ ATIO2                // Se activa la salida digital 1. No se activa la 2.
← OK
→ ATIO3                // Se activan las salidas digitales 1 y 2.
← OK
→ ATM0=0               // Se fija PWM0 con valor 0.
← OK
→ ATM0200              // Se fija el PWM0 con valor 0x200 que equivale a la mitad de la salida.
                       // En la salida obtenemos un valor de tensión continua de 1,64 V
← OK
→ ATM03FF              // Se fija el PWM0 con valor 0x3FF que es el máximo de salida.
                       // Se obtiene  3,28V ~ 3,29 V = Vcc.
← OK
```

Ahora ya se ha comprobado la forma de activar salidas digitales y analógicas en modo local mediante comandos AT, teniendo el módulo directamente conectado al PC mediante el puerto serie. Ahora se va a utilizar el API para enviar estos comandos de forma remota, a través de otro XBee que si ha de estar conectado al PC.

En este primer caso se usa un módulo XBee como nodo actuador y otro como estación base, que será el encargado de enviar la información al receptor. Se realizará sin entrar ninguno en modo de bajo consumo (*Sleep*). La configuración a utilizar en los módulos es mostrada en la Tabla 4.9.

	Módulo 1 Coordinador	**Módulo 2 Disp. Final 1**	**Descripción**
MY	1111	5001	Dirección corta del módulo en la red.
ID	3332	3332	PAN ID. Igual en el Coordinador y el EndDevice.
SM	0	0	Módulos no entran en modo Sleep.
CE	1	0	Coordinador habilitado.
A1	0	0110b	En Disp. Final se configura para asociarse.
A2	011b	0	El Coord. Se configura para asociar a módulos con su mismo PanId.
AP	1	1	Habilitación modo API.
P0	--	2	Configuramos el pin como Salida PWM.
D1	--	4	Configuramos la DIO1 como salida digitales, con valor bajo por defecto.
D2	--	5	Configuramos la DIO2 como salida digitales, con valor alto por defecto.
T1,T2	--	0x64 (10 s)	Valor de validez de la salida digital, cuando expira vuelve a su valor por defecto. Máximo 25,5 s.
PT	--	0x64 (10 s)	Valor de validez de la salida PWM0, cuando expira vuelve a 0. Máximo 25,5 s.

Tabla 4.9: Configuración red con XBee para usar salidas mediante comando remotos

Para enviar los comandos AT de forma remota se ha de usar la trama de API de petición de comando remoto, cuya estructura es la mostrada en la Ilustración 4.15.

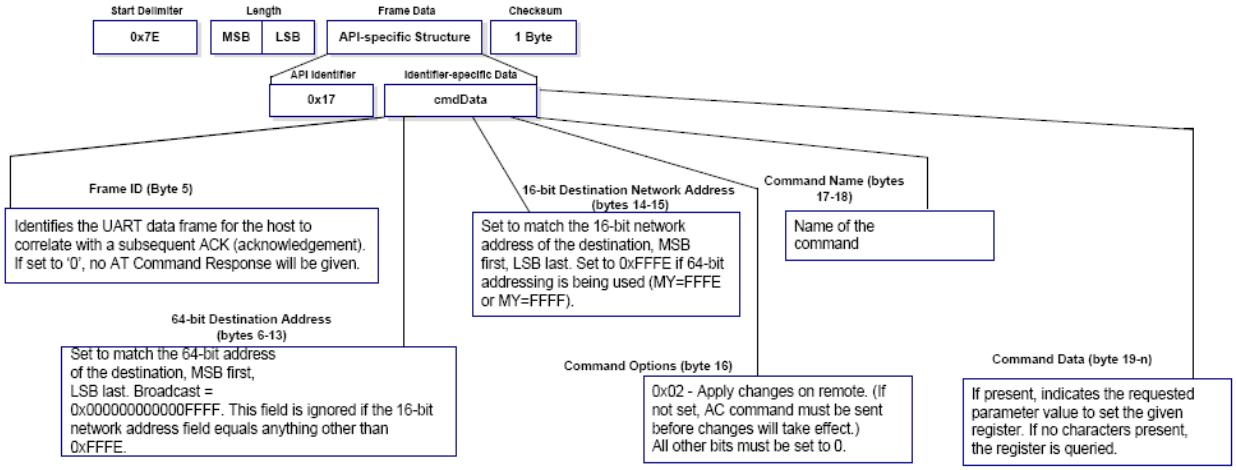


Ilustración 4.15: Estructura de trama Petición de comando remoto AT

Pare enviar la petición de comando remoto se pude especificar la dirección de 64 bits o la de 16 bits. En el caso de usar la de 64 bits, en el campo de la dirección de 16 se ha de poner el valor 0xFFE. Para usar la dirección de 64 bits, se ha de poner en el campo de la dirección de 16 bits cualquier dirección diferente a la anterior. Según el manual lo que haya escrito en la dirección de 64 bits será ignorado. En la práctica se ha comprobado que el módulo no envía las peticiones usando la dirección corta de 16 bits si en el campo de la dirección de 64 bis tiene algo diferente a todo 0.

A continuación se especifica la trama necesaria para cambiar el valor de las dos salidas digitales a estado alto, así la trama queda:

7E 00 10 17 01 00 00 00 00 00 00 00 00 50 01 02 69 6F 60 B6

El significado de los bytes de la trama recibida es:

0x7E	: Identificador.
0x0010	: Tamaño de la trama.
0x 17	: Identificador de comando AT Remoto.
0x0000000000000000	: Dirección de 64 bits todo a 0.
0x50 01	: Dirección de 16 bits del dispositivo final que ejecutara el comando.
0x02	: Opción de comando, para que se ejecute inmediatamente.
0x 69 6F	: Comando AT IO.
0x06	: Parámetros del comando IO. 6 → 0000 0110 Activos salidas 1 y 2
0x B6	: Checksum

Como respuesta se obtiene la activación a nivel alto las salidas digitales 1 y 2 y el XBee conectado al PC envía la siguiente trama de confirmación por el puerto serie:

7E 00 0F 97 01 00 13 A2 00 40 52 46 75 50 01 69 6F 00 3C

El significado de los bytes de la trama recibida es:

0x 7E	: Identificador inicio de trama.
0x 00 0F	: Tamaño de la trama.
0x 97	: Identificador de la trama, **0x97= Respuesta de comando remoto AT.**
0x 00 13 A2 00 40 52 46 75	: Dirección hardware del módulo emisor de la trama. (64 bits)
0x 50 01	: Dirección corta del módulo emisor de esta trama. (**5001**)
0x 69 6F	: Comando AT al que responde. **Comando IO**
0x 00	: Estado de la ejecución del comando remoto OO = **Todo OK.**
0x 3C	: Checksum de la trama.

Si se envía un comando remoto y el emisor no consigue entregarlo o recibir la confirmación de que ha sido entregado, pasado un tiempo, se recibe una trama de respuesta de comando remoto, con el valor de 0x04 en el campo de Estado de la ejecución, indicando así que el mensaje no ha llegado al receptor.

Para activar la salida de PWM se hace del mismo modo visto anteriormente, pero con la siguiente trama que envía el comando AT M0 con su parámetro de nivel de salida al valor hexadecimal de 0x384, muy cercano al valor máximo que es 3FF.

Trama: 7E 00 11 17 05 00 00 00 00 00 00 00 00 50 01 02 4D 30 03 84 8C

El significado de los bytes de la trama enviada es:

0x 7E	: Identificador de inicio de la trama.
0x 0011	: Longitud de la trama.
0x 17	: Identificador de comando remoto AT.
0x 05	: Identificador de la trama.
0x 00 00 00 00 00 00 00 00	: Dirección de 64 bits del receptor. Se deja todo a 0 para usar el : direccionamiento de 16 bits.
0x 50 01	: Dirección corta (16 bits) del receptor.
0x 02	: Byte de opciones, ejecución inmediata del comando AT.
0x 4D 30	: Comando AT en ASCII. MO, comando fijación valor PWM0.
0x 03 84	: Parámetro del comando M0 (0x384), valor de la salida entre 0 y 3FF.
0x 8C	: Checksum de la trama.

Una vez recibida la trama por el XBee remoto configurado como dispositivo final (nodo actuador), este fija su salida al valor indicado y envía la trama de confirmación al XBee base, que la transmite al PC por el puerto serie. La trama recibida es la siguiente:

7E 00 0F 97 05 00 13 A2 00 40 52 46 75 50 01 4D 30 00 93

El significado de los bytes de la trama de respuesta recibida es:

0x 7E	: Identificador inicio de trama.
0x 00 0F	: Tamaño de la trama.
0x 97	: Identificador de la trama, 0x97= **Respuesta de comando remoto AT**.
0x 00 13 A2 00 40 52 46 75	: Dirección hardware del módulo emisor de la trama. (64 bits)
0x 50 01	: Dirección corta del módulo emisor de esta trama. (**5001**)
0x 4D 30	: Comando AT respondido en ASCII. **Comando M0, valor PWM0.**
0x 00	: Estado de la ejecución del comando remoto OO = **Todo OK**.
0x 93	: Checksum de la trama.

En el XBee actuando de dispositivo final, en su salida PWM0 se tiene un valor en tensión de 2,89 V.

Para comprobar que este valor medido en la salida del PWM0 efectivamente corresponde al valor que se ha enviado, se realizan los cálculos de la salida teórica a obtener. El valor de 0x384 es la 0,88 parte de 3FF, por tanto, si en 3FF ha de dar como salida la tensión de alimentación (3,3 V) la salida teórica ha de ser: 2,904 V. Se observa que la salida teórica corresponde con la salida real. La pequeña diferencia es debida a que la alimentación realmente tiene un valor 3,28 V.

4.3 Conclusiones

En este capítulo se han realizado varias pruebas para comprobar el funcionamiento de los diferentes modos de funcionamiento del XBee. Además, se ha documentado la configuración necesaria de los XBee para usar cada una de ellas. Incialmente se ha explicado la configuración más simple del modo transparente, el I/O Passing, que tiene aplicaciones directas de reemplazo de cableado. También se ha descrito el modo de bajo consumo, que es fundamental para mantener la vida de las baterías, y por último, el uso del API, para la configuración remota de los módulos, así como el uso de las entradas y salidas del XBee.

En la realización de las pruebas se ha podido observar, nociones a tener en cuenta del manejo de los XBee, como por ejemplo que si se configura el valor del ciclo de Sleep a un valor muy pequeño, es muy difícil reconfigurar el módulo ya que el tiempo que pasa

activo es muy pequeño. Al igual se ha comprobado que para usar el direccionamiento de 16 bits para enviar comandos remotos, el campo de dirección de 64 bits ha de ser todo 0, pues en otro caso no se envía el paquete.

En el próximo capítulo se usarán las configuraciones descritas para el diseño del caso de estudio, donde se plantea un problema y su resolución, utilizando para ello las capacidades de los módulos XBee.

4.4 Bibliografía

[1] XBee manual de usuario. Disponible on-line en: http://ftp1.digi.com/support/-documentation/90000982_B.pdf

Capítulo 5

Caso de Estudio

5.1 Introducción

En este capítulo se describe el caso de estudio propuesto para validar el funcionamiento de una arquitectura de red basada en módulos XBee. Concretamente, el caso de estudio consiste en el control de un proceso industrial, que se describe en el siguiente apartado. A grandes rasgos, el sistema está compuesto por sensores industriales, que envían su señal a través de módulos XBee hacia un XBee conectado a un PC, qué actúa como nodo sumidero de la red. Éste último traslada las lecturas a un programa informático que se encarga de calcular y generar las órdenes de proceso, las cuales se envían a través de la red inalámbrica hasta los módulos inalámbricos XBee, qué están conectados a los actuadores industriales.

5.1.1 Caso Práctico

En la planta de exprimido de la empresa Confridul en Jumilla, debido al crecimiento que ésta última ha experimentado estos últimos años, el tamaño actual de la planta tratadora de aguas sucias provenientes de la limpieza de la fruta no es suficiente y su actual ubicación no permite una ampliación. Para la solución de este problema, se han adquirido unos terrenos fuera de las actuales instalaciones y se ha montado allí el nuevo centro de tratado de las aguas sucias. Este centro de tratado consiste en dos reactores biológicos, consistentes en dos tanques abiertos de gran capacidad. En el primero se airea

el agua sucia para aumentar la cantidad de oxígeno en la disolución, para que así las bacterias aerobias puedan reproducirse con facilidad y consumir la materia orgánica. En el segundo tanque es donde se almacena el agua durante un tiempo, típicamente 4 horas, para que decante toda la materia sólida y que las bacterias anaeróbicas que están presentes en la capa de fango del fondo del tanque terminen el proceso con la destrucción de la materia orgánica todavía presente, quedando el agua así apta para su reutilización. Esta agua ya tratada se va utilizando según las necesidades y sólo es necesario extraerla por la salida de servicio, que está colocada dos tercios de la altura del tanque, con objeto de extraer sólo agua ya decantada y que no se pierda la capa de fangos donde viven las bacterias anaeróbicas.

De la instalación antigua se ha reutilizado el tanque de aguas sucias, que ahora pasa a ser el primer tanque colector del agua. Éste es el que abastece a través de una bomba a la instalación nueva.

Con el nuevo proceso de aireado que se ha incluido, es necesario que el nivel del agua en el primer tanque sea siempre constante ya que la aireación no es regulable. Por ello, si el nivel de este tanque es demasiado bajo, los aireadores hacen que las aguas salpiquen y se agiten demasiado. Esto produce inestabilidades en el tanque, al igual que si el nivel es demasiado elevado el agua rebosa.

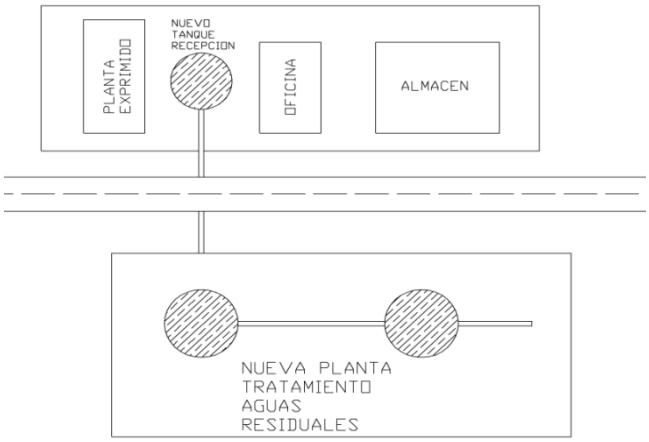


Ilustración 5.1: Plano representativo de las instalaciones

Por tanto, lo que se ha de controlar es el nivel de aportación al primer tanque, que ha de mantener su nivel y que se vacía según las necesidades de consumo de agua tratada del tanque 2. El llenado se realiza mediante la bomba de las instalaciones viejas, colocada a la salida del tanque reutilizado como recolector de las aguas sucias de la planta. La distancia entre la planta de tratado antigua y las nuevas instalaciones es de 800 metros. En la Ilustración 5.1 se puede observar un plano representativo de la situación de las instalaciones. Puesto que, por medio de ambas pasa una carretera, instalar el cableado requerido por un sistema de control tradicional supondría un gran coste. Por este motivo,

se ha decidido la instalación de la medición de caudal del tanque 1 de la nueva instalación y el accionamiento de la bomba de la plataforma antigua mediante módulos XBee, que estarán controlados desde un PC situado en la oficina del responsable de la planta de exprimido.

5.2 Descripción del Hardware

La bomba del tanque de recolección de agua situada en la planta antigua está comandada por un variador de frecuenciaV-1000 [1] de la marca Omron, que toma su referencia de velocidad a través de su entrada analógica A1, con escala 0-10 V.

Para conocer el nivel de agua del tanque de aireación se usa un sensor de ultrasonidos que proporciona una salida en el rango de 4-20mA. Para ello, se usa la sonda de nivel E+H FMU41 [2], con capacidad de medición de hasta 8 metros de distancia.

El responsable de exprimido debe de saber en todo momento el nivel del tanque de aireación, para poder detener el proceso en caso necesario, así como que quede un registro de dicho nivel para la confirmación del correcto tratamiento de las aguas sucias.

Se propone utilizar módulos XBee-Pro, que trabajando bajo el estándar IEEE 802.15.4 en la frecuencia de 2.4GHz, van a permitir conocer el valor de esas señales en el PC de la sala de control, así como actuar sobre ellas.

Se dispondrá de un módulo en la bomba del tanque de recolección, otro en el tanque de aireación y finalmente, otro conectado mediante el puerto serie a un PC en la sala de control.

Se identifican a estos módulos como:

- Módulo – Sensor. (El situado en el tanque de aireación)
- Módulo – Actuador. (El situado en la bomba de recolección)
- Módulo – Base. (El conectado al PC)

5.2.1 Módulo Sensor

Para la conexión del módulo sensor al transmisor de nivel se utiliza una de las 5 entradas de convertidor analógico digital que posee el XBee, así como un circuito acondicionador que permita tener el rango de entrada entre 0 y 3,3 V, a partir de la salida de 4-20mA, que proporciona el sensor de distancia por ultrasonidos.

Se propone el circuito acondicionador mostrado en la Ilustración 5.2 encargado de recibir la intensidad proveniente del sensor, el cual tiene una escala de 4-20mA, y adaptarla a una tensión adecuada para las entradas analógicas de los módulos, que aceptan desde 0 a 3,3V. Para ello, mediante la resistencia de precisión de 165 ohmios conectada a tierra, se tiene que para la intensidad máxima que proporciona el 100% de la señal, los 20 mA, se obtienen 3,3V. Esta tensión es la entrada a un seguidor de tensión realizado usando un amplificador operacional LM358[3] con objeto de desacoplar la salida y la entrada. La salida del seguidor ya se puede conectar directamente a la entrada analógica del módulo XBee. A la salida del seguidor de tensión también se ha colocado un diodo Zenner de 3,3V (D1N4728)[4], para proteger contra una sobretensión a las entradas del módulo XBee.

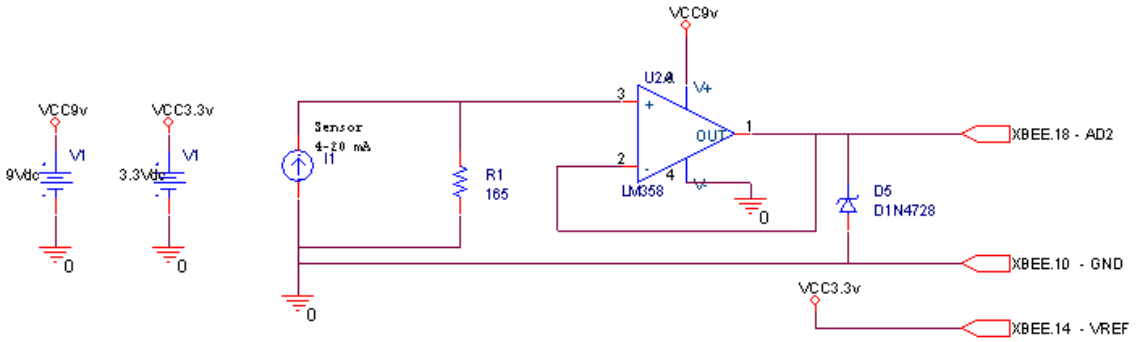


Ilustración 5.2: Circuito acondicionador del módulo sensor

La resistencia que se usa para obtener la caída de tensión que se suministra al seguidor de tensión, está dentro de los valores de las utilizadas por las tarjetas A/D de los autómatas programables comerciales. Así, por ejemplo la tarjeta de entrada analógica del PLC Omron C200h-AD002, tiene un valor ente 100 y 250 Ω[5].

Diseño de la placa adaptadora para XBee y circuito acondicionador

Se ha diseñado una placa de circuito electrónico que permite conectar hasta dos señales de corriente en el rango 4-20mA al módulo XBee, aunque en el proyecto sólo sea necesaria una entrada. Se ha diseñado con dos para la posible ampliación futura de la monitorización de la temperatura o cualquier otro parámetro del proceso. El diseño consiste en un zócalo para conectar el XBee, dos terminales de conexión para conectar dos sensores analógicos con salida 4-20mA, y una toma de alimentación, de donde se obtienen la tensión de 3.3V de alimentación del XBee, a través de un regulador lineal LM317[6].

En la placa se ha incluido un diodo conectado a la salida DIO5 del XBee para que sirva de indicativo de su estado de asociación (Cuando está asociado con el coordinador parpadea a un ritmo de 5 veces por segundo). El esquema definitivo en el software Orcad Capture es el mostrado en la Ilustración 5.3. En el diseño también se ha incluido un botón de reset para resetar el XBee y dos conectores (IDC1 en el esquemático) para el puerto

serie, es decir para los pines D1 y D2 del XBee, para así poder conectarlo a un PC y reprogramarlo en caso necesario.

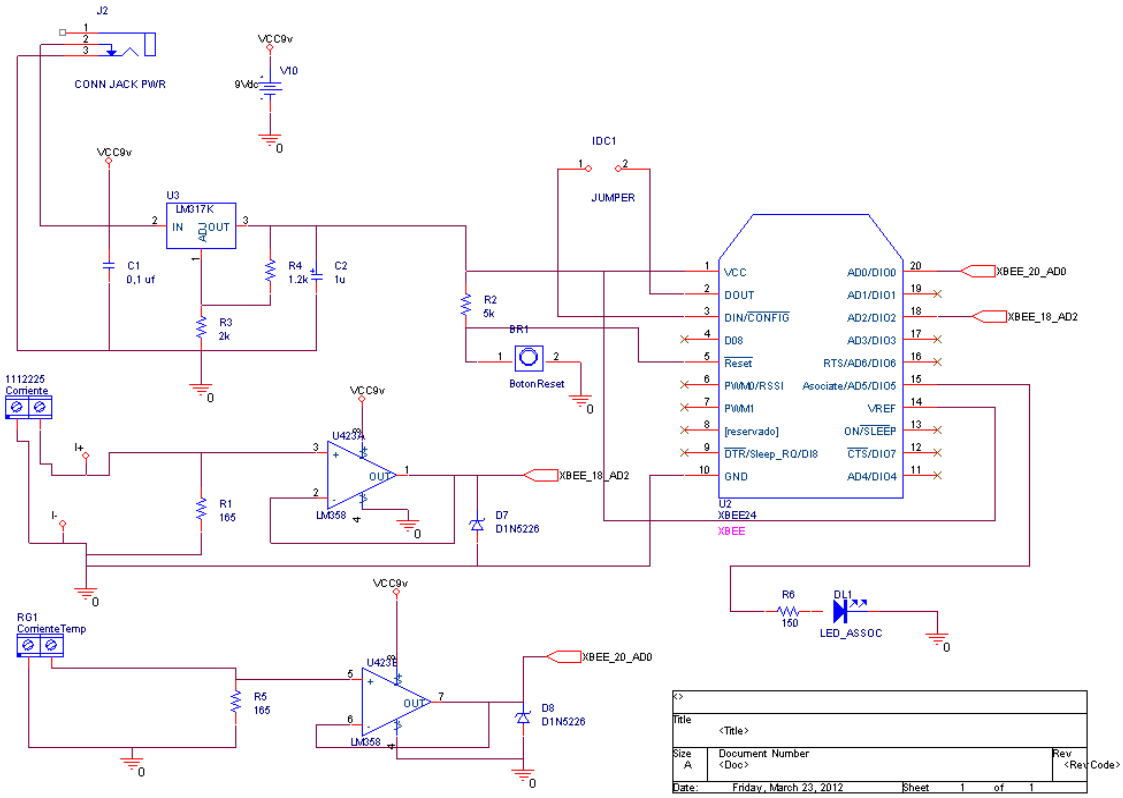


Ilustración 5.3: Diagrama completo adaptador XBee sensor

Una vez realizado el diseño del esquemático final, se procede a diseñar la PCB de la placa final, usando Layout, para su posterior fabricación. Para obtener la mayor simplicidad se ha diseñado a una sola cara, con componentes de agujero pasante. El resultado se muestra en la Ilustración 5.4.

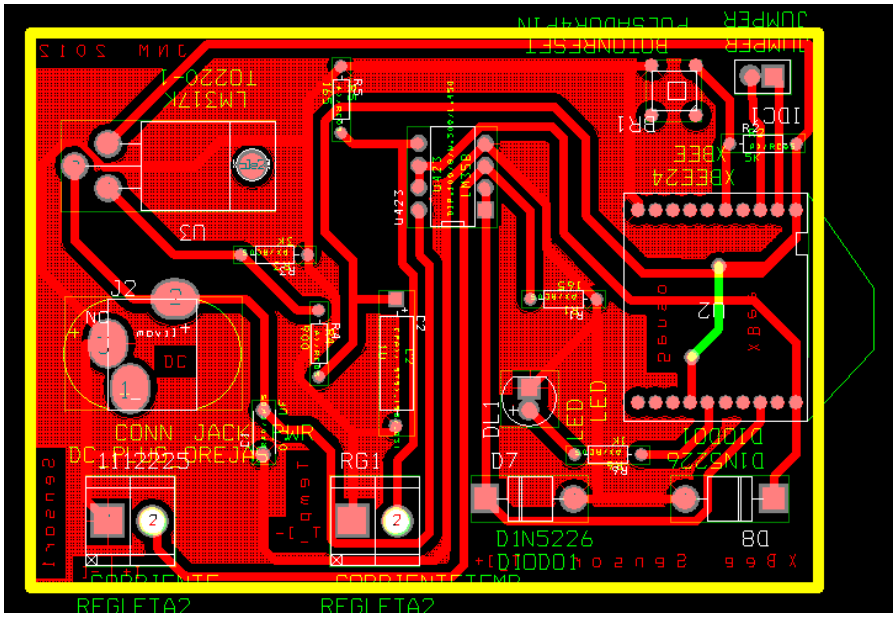


Ilustración 5.4: PCB del adaptador XBee sensor

Una vez terminado el diseño, se pasa a la realización de la PCB, donde una vez obtenidas las pistas de cobre, se mecanizan los agujeros y se sueldan los componentes. El resultado final, a falta de colocar el XBee en su zócalo se observa en la Ilustración 5.5.

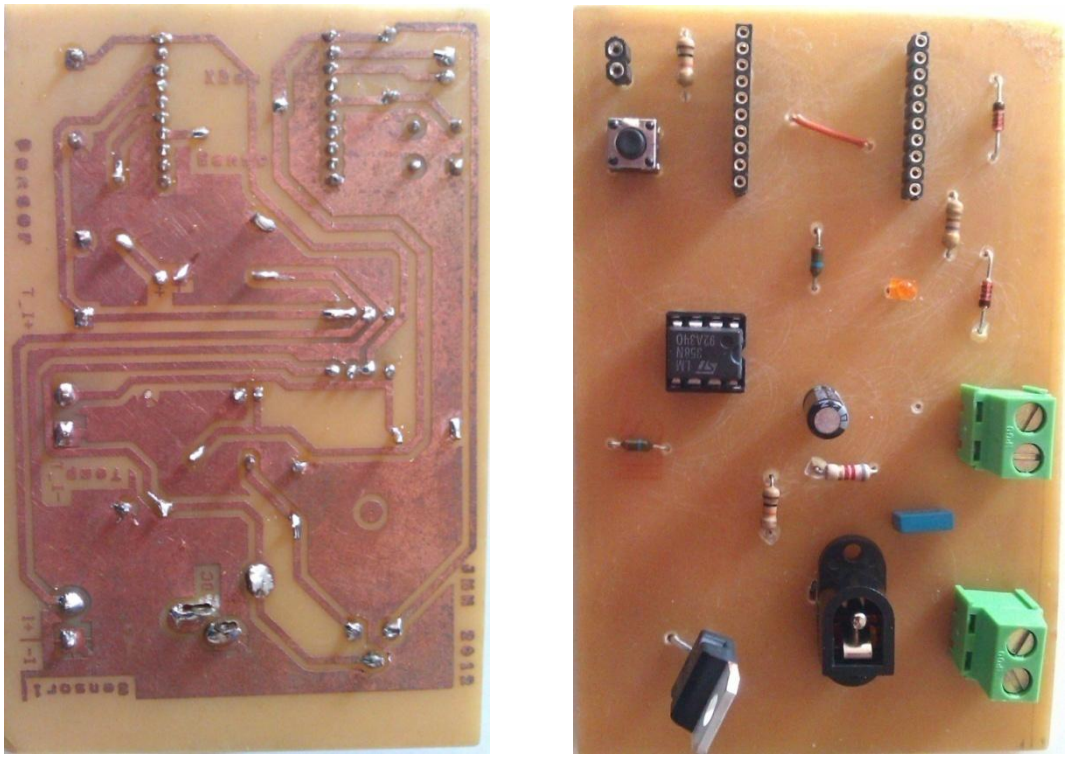


Ilustración 5.5: Adaptador XBee sensor terminado

5.2.2 Módulo Actuador

Para la conexión del módulo actuador al variador de la bomba se utiliza un circuito de acondicionamiento (ver Ilustración 5.6), que a partir de la tensión de salida del PWM del XBee, en el rango de 0-3,3V, una vez filtrado, permite obtener una salida analógica con valores comprendidos entre 0 y 10 V, que es la entrada de referencia al variador de frecuencia.

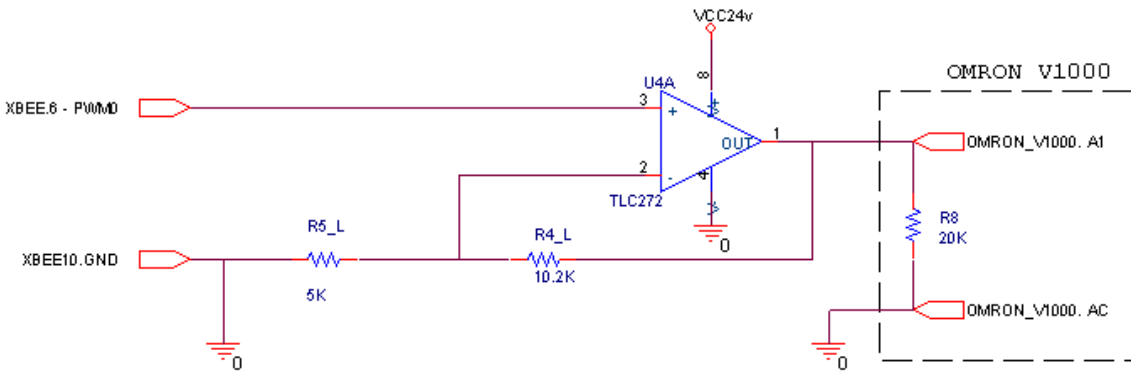


Ilustración 5.6: Circuito acondicionador del módulo actuador

La resistencia de 20kΩ de la Ilustración 5.6 simula la conexión al variador. Este valor se ha obtenido de la especificación de la impedancia de la entrada analógica A1 del variador V1000 de Omron[1]. En el Anexo I, se encuentra la configuración de los parámetros del variador V1000, necesaria para la aplicación.

El circuito acondicionador del módulo actuador se encarga de proporcionar una salida de 0-10V a partir de las salidas 0-3,3V PWM de los módulos XBee. Para ello, se usa un amplificador operacional en configuración no inversora con una ganancia de 3,04 V/V, lo que hace que la salida del PWM máxima de 3,3V se amplifique a 3,3V* (2,04+1) = 10,032V. Se usa el amplificador operacional de potencia L272, que puede proporcionar una intensidad máxima de 1 A. Para conseguir la relación de amplificación deseada se usan dos resistencias de 5k y 10,2 k, respectivamente, que ofrecen el valor de 3,02 de ganancia del amplificador operacional en la configuración de no inversor.

Diseño de la placa actuador para XBee

Para realizar la adaptación de niveles de tensión entre la salida del XBee y la entrada del actuador industrial, en este caso el variador de frecuencia que comanda la bomba, se ha diseñado el circuito mostrado en la Ilustración 5.7.

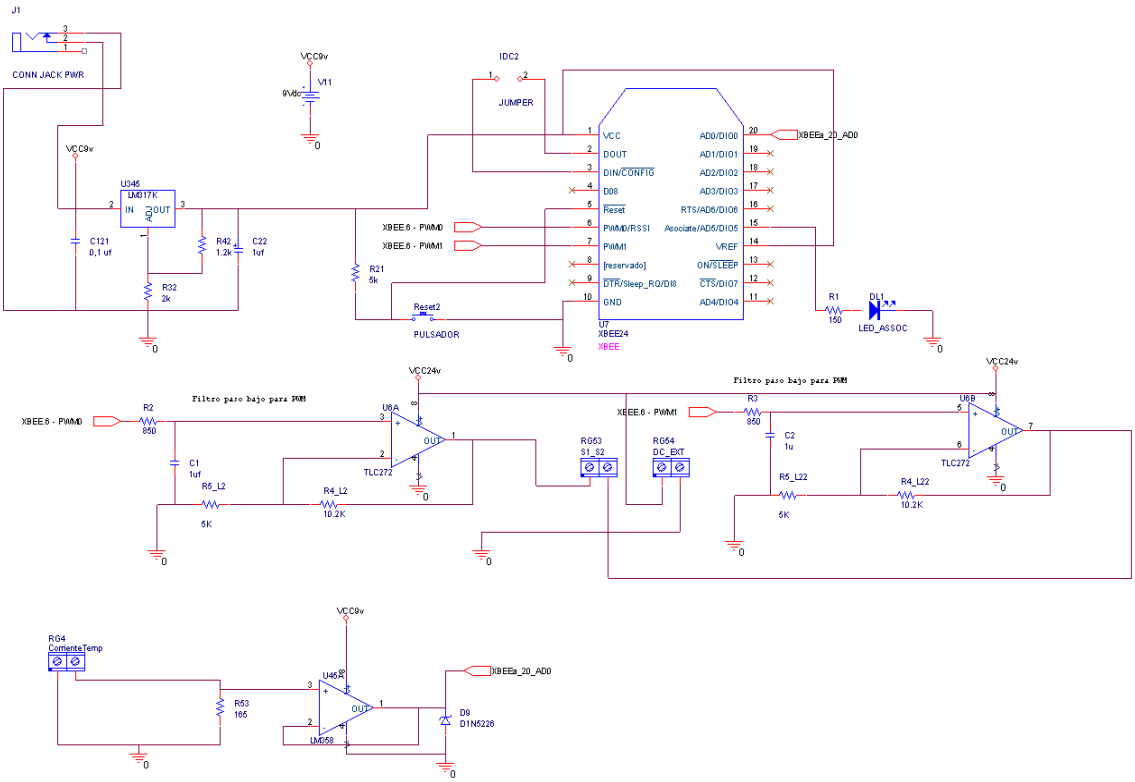


Ilustración 5.7: Diagrama de placa adaptadora señales XBee actuador

En el diseño se incluye un zócalo para el XBee, un regulador LM317[6] para obtener los 3,3V necesarios para la alimentación del XBee, a partir de la entrada que se

obtiene del conector de tipo Jack. Como elemento principal para adaptar los niveles de tensión se usa el circuito acondicionador visto en el apartado anterior, que está basado en el amplificador operacional de potencia L272[7]. A este amplificador se le conecta la señal que sale de los pines PWM0 y el PMW1 del XBee, pasando por un filtro paso-bajo, formado por una resistencia de 850 Ω y un condensador de 1 µF. Con esto se filtra la frecuencia del PWM del XBee, que es de 15,625 kHz.

La salida del amplificador L272 hacia el exterior se proporciona a través de dos conectores dobles, el primero tiene dos salidas, 0-10V y el segundo es la alimentación de potencia del amplificador operacional, que debe ser en el rango 12V - 30V. En este caso no es necesaria alimentación simétrica.

Una vez realizado el diseño se pasa a diseñar la PCB mediante el programa de diseño Orcad Layout. Por simplicidad en la fabricación se ha diseñado con pistas a una sola cara y con componentes de agujeros pasantes. El resultado es el layout mostrado en la Ilustración 5.8.

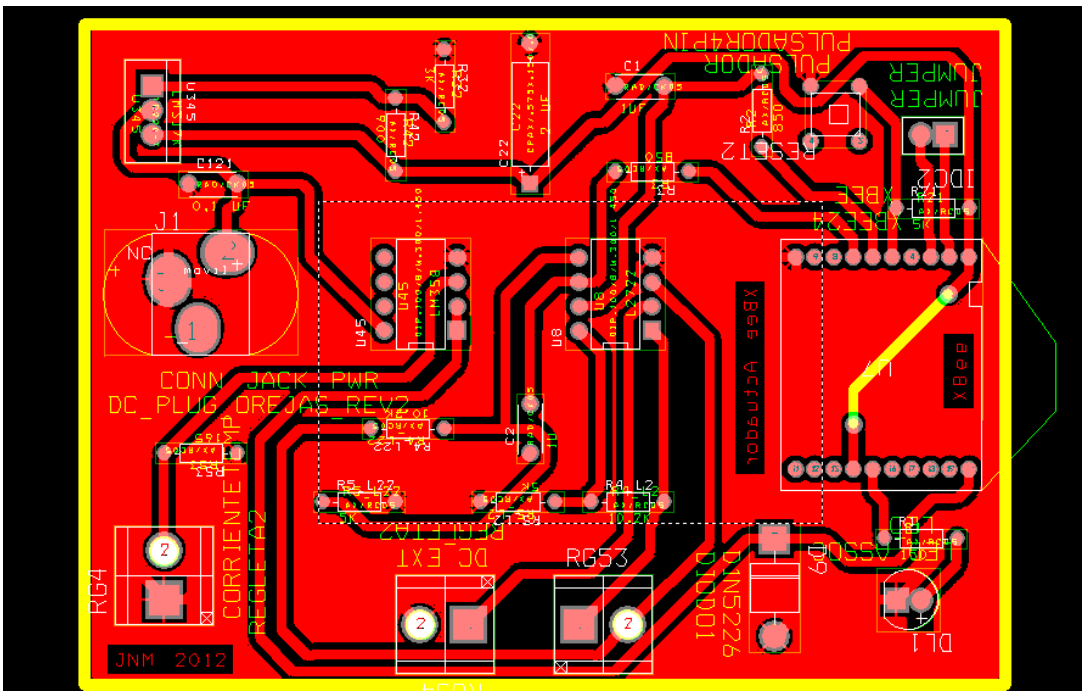


Ilustración 5.8: PCB circuito actuador para XBee

Una vez montado el circuito queda como se muestra en la Ilustración 5.9.

Como en el diseño de la placa sensor, se ha incluido un LED conectado al Pin de D5, para indicar el estado de asociación del XBee, un botón para resetear el XBee y dos conectores para sus pines D1, D2, para permitir su reprogramación mediante el puerto serie. La alimentación de la placa se realiza por medio de un conector DC, hacia un regulador lineal LM317, con objeto de obtener 3,3V para alimentar el módulo XBee. La alimentación en el conector DC ha de ser 9 V de corriente continua.

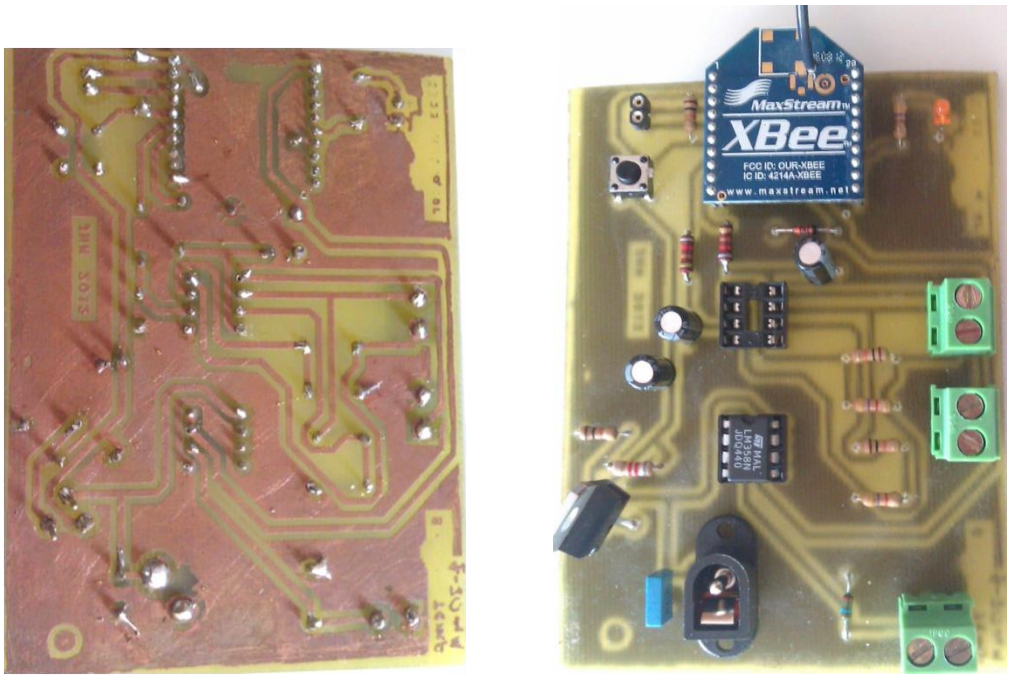


Ilustración 5.9: PCB realizado con componentes montados

5.2.3 Estación Base

Para realizar la conexión del módulo base al PC se puede usar una placa de desarrollo que lleve incorporado un puerto serie, como pueden ser las XBIB-R-DEV, o diseñar una pequeña placa con el circuito mostrado en la Ilustración 5.10, que acondiciona las señales del puerto serie del XBee a las del puerto serie del PC mediante el circuito integrado Max3232 [8] y algunos componentes discretos.

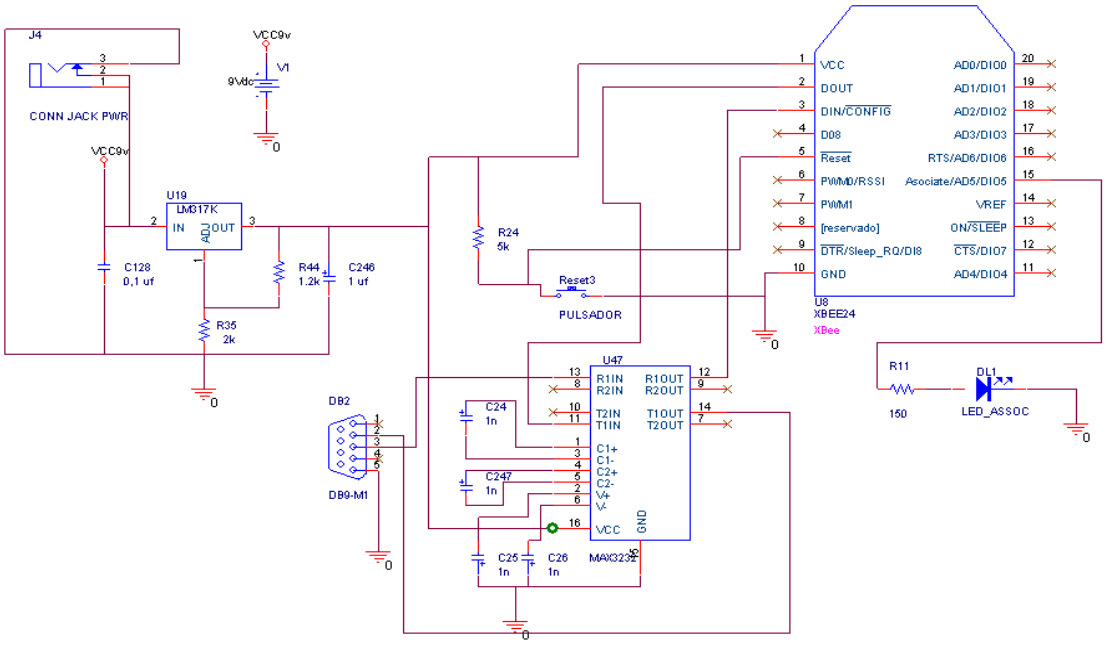


Ilustración 5.10: Esquemático de la estación base del módulo XBee base

En este diseño se han adaptado las señales de la UART del XBee para poder conectarlo a un puerto serie estándar de PC mediante un conector DB9 hembra. Para ello, además de usar el CI Max2322 [8], se ha incluido un conector DC por el que se alimenta un regulador LM317[6] ajustado para proporcionar los 3,3V de alimentación del XBee. También se incluyen un botón para resetear el XBee y un LED en el Pin DIO5, para poder conocer el estado de asociación del XBee.

Para su implementación se realiza el diseño de la PCB mediante el software Orcad Layout. El diseño se muestra en la Ilustración 5.11.

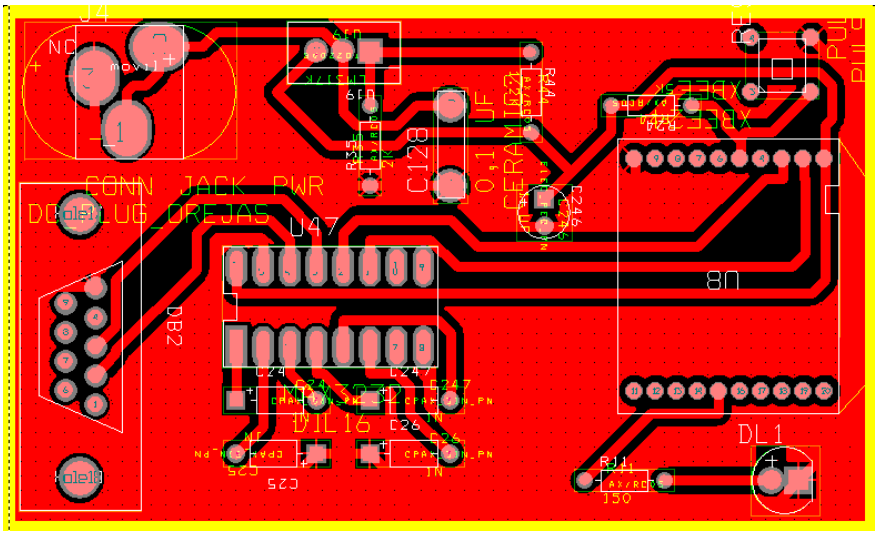


Ilustración 5.11: Layout del diseño de estación base

Finalmente, la placa realizada y montada se muestra en la Ilustración 5.12.

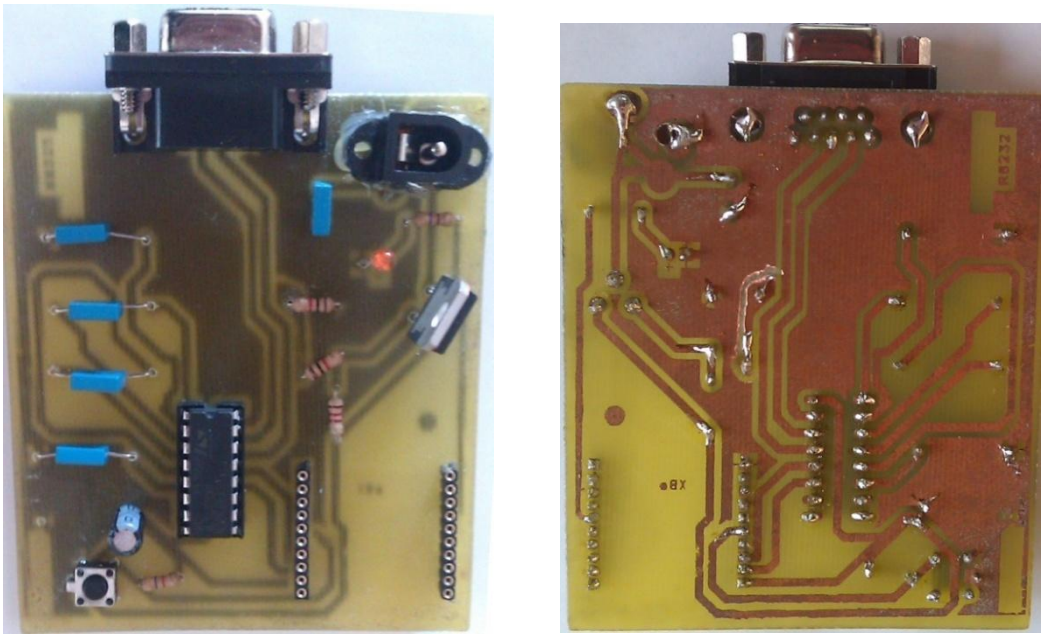


Ilustración 5.12: Placa montada para XBee base

5.3 Descripción del Software

En este apartado se describen la configuraciones necesarias que se realizan sobre los XBee para el desarrollo de la aplicación de este caso de estudio. Con estos módulos se tienen distintas posibilidades de comunicación con la estación base, como el IO Passing, modo transparente, uso de comandos AT y comandos API. En este caso, se ha elegido la comunicación basada en el uso del API, ya que permite el total uso de las funciones del XBee, en las que se destacan la lectura de las entradas analógicas y digitales periódicamente, el envió de los paquetes de datos y el envió de comandos remotos para actualizar su salida de PWM. Este último es el que se usará como salida del módulo actuador.

5.3.1 Configuración de los Módulos XBee

Configuración del módulo sensor

En la Tabla 5.1 se detalla la configuración a realizar en el XBee del módulo sensor, para que trabajando con modo *Sleep* y comunicaciones indirectas, sea capaz de trasmitir la señal analógica recogida en su AD2 (la señal de nivel del tanque), hasta la estación base, que será su coordinador. Al estar configurado IR (*Sample Rate*), y SM (*SleepMode*), el módulo no entrará en modo de bajo consumo hasta que se recojan las 2 muestras indicadas por IT (*Samples before TX*), que son las muestras que se han de tener antes de enviar el paquete por la radio.

	Módulo sensor	Descripción
MY	5001	Dirección corta del módulo en la red.
DL	1111	Dirección de destino. Dirección del Coordinador.
ID	3332	PAN ID. Identificador de la Red creada por el coordinador.
SM	4	Se configura para que pase a modo Sleep de forma periódica.
SP	0x1f4	Periodo de *Sleep* 5s. Se define el tiempo entre muestras, que será el tiempo que el módulo permanecerá en modo *Sleep*.
ST	0x200	Tiempo sin actividad antes de ir a sleep. (0,5s)
CE	0	Se le indica que actuará de dispositivo final.
A1	1100b	Se configura como dispositivo final para asociarse y se le indica el tipo de comunicación como indirecta. (bit 3)
A2	0	Este módulo no es coordinador. Debe de estar a 0.
AP	1	Habilitación modo API. Para poder leer el valor de su entrada en una trama API.
IR	0x1f4	Periodo de muestreo 0,5 s. Tiempo entre las dos muestras que se envían en el mismo paquete.
IT	2	Se recogen dos muestras para enviarlas seguidas. Esto se hacer para evitar falsas lecturas.
Dn	D2=2	Se configura la entrada AD2 (pin18) como entrada analógica.

Tabla 5.1: Configuración del módulo sensor

Después de enviar el paquete y pasado el tiempo de ST (*Time before Sleep*, tiempo sin actividad antes de pasar a modo Sleep), el XBee entra en modo de bajo consumo (*Sleep*). El XBee envía sus tramas con los datos de las entradas a la dirección configurada mediante DL (*Destination address Low*) y DH (*Destination address High),* siendo DH=0*,* para usar el direccionamiento de 16 bits, y DL=1111 la dirección corta de 16 bits, de la estación base. El resto de parámetros que no aparecen en la Tabla 5.1 se han dejado al valor por defecto.

Configuración del módulo actuador

En la Tabla 5.2 se detalla la configuración a realizar sobre el módulo XBee que incorpora la placa módulo actuador. El objetivo es que sea capaz de recibir paquetes desde el coordinador con la información de la intensidad de señal a la que ha de ajustar su PWM0, que define la consigna de velocidad para la bomba.

Para que los paquetes sean recibidos por este módulo, deben ir dirigidos a la dirección de 64 bits del módulo, ya que al asociarse el parámetro MY cambia a 0xFFFE, y no puede usarse el direccionamiento de 16 bits. Este módulo no puede entrar en modo *Sleep* mientras sea necesario mantener la referencia al variador. Para permitir que en los periodos que no sea necesario una actuación sobre la instalación el XBee cambie a modo de bajo consumo, es necesario que el coordinador o el programa que le envía datos esté preparado para no enviar paquetes en caso de que la salida a usar sea cero, es decir, si la bomba debe de estar parada. Una vez se haya recibido el primer paquete indicando tal situación, se deberá dejar de enviar paquetes al módulo para así permitir que este entre en modo suspendido.

	Módulo Sensor	**Descripción**
MY	5002	Dirección corta del módulo en la red. Antes de asociarse.
DL	1111	Dirección de destino. Dirección del Coordinador.
ID	3332	PAN ID. Identificador de la Red creada por el coordinador.
SM	4	Se configura para que pase a modo *Sleep* de forma periódica.
SP	0x1f4	Periodo de *Sleep* 5s. Se define el tiempo entre comprobaciones de activación de su salida.
ST	0x2710	Tiempo sin actividad antes de ir a Sleep de 10 s. Este tiempo ha de ser mayor que el tiempo de envío de tramas por el coordinador, para que mientras esté enviando datos el coordinador, este módulo no se vaya a modo suspendido. Una vez la referencia de salida sea 0, el coordinador no debe de enviar comandos de refresco de salida de PWM, para que el módulo pueda ir a modo Sleep pasado el periodo ST definido.
CE	0	Se le indica que actuará de dispositivo final.
A1	1100b	Se configura como dispositivo final para asociarse y se le indica el tipo de comunicación como indirecta (bit 3).
A2	0	Este módulo es dispositivo final debe de estar a 0.
AP	1	Habilitación modo API. Para poder modificar el valor de su salida PWM en remoto desde la estación base conectada al PC mediante una trama API.
P0	2	Se configura el pin6 como Salida PWM0.

Tabla 5.2: Configuración del módulo actuador

El parámetro ST=0x2710 (10 segundos), indica que ha de estar 10s sin recibir ningún paquete por su radio antes de pasar a modo de bajo consumo. Una vez entre en este modo, tardará 5 segundos en comprobar si tiene nuevas peticiones, el coordinador usa con él transmisiones indirectas. El periodo de envío de paquetes con información del valor de la salida de PWM ha de ser inferior a 10 segundos, para que mientras la bomba necesite estar comandada, el módulo no se pase a modo *Sleep*, lo que desconecta la salida de PWM y se queda a un valor de 0, en este periodo el coordinador usa trasmisiones directas.

Actualmente el software de envío de tramas desde el PC está programado para enviar una trama de actualización del valor del PWM cada 5 segundos. Los demás parámetros no especificados en la Tabla 5.2 se han dejado a su valor por defecto.

Configuración del módulo estación base

En la Tabla 5.3 se detalla la configuración a realizar sobre el módulo XBee utilizado en la estación base. Este módulo, estando conectado por el puerto serie al PC (ó USB a través de un conversor USB a Rs232), se encargará de recibir los paquetes del módulo sensor, así como de enviar las tramas con la salida de control hacia el módulo actuador. Este XBee tiene que estar en el alcance de la radio de los otros dos XBee.

Este módulo será el encargado de suministrar al programa de control, que se encuentra ejecutándose en el PC, la información del nivel del tanque, y en función de esta, el programara responderá con una salida de control, la velocidad de la bomba, que será transmitida hasta el módulo actuador colocado en la bomba. Las estación base usa comunicaciones indirectas mientras los módulos están en *Sleep*, cuando detecta que están activos usa con ellos comunicaciones directas durante el tiempo marcado en ST (*Time before Sleep*). Los demás parámetros no especificados en la Tabla 5.3 se usa su valor por defecto.

	Módulo Sensor	**Descripción**
MY	1111	Dirección corta del coordinador en la red.
DL	0xFFFF	Dirección de destino. El coordinador enviará sus paquetes a todos los dispositivos finales. Esto no se aplica a las tramas API que llevan su propia dirección de destino.
ID	3332	PAN ID. Esta prefijada para la red que se crea.
CE	1	Se habilita para ser el coordinador de la red.
A1	0	Como es coordinador no tiene efecto y se deja a 0.
A2	100b	El Coordinador se configura para asociar a módulos con su mismo PanId.
AP	1	Habilitación modo API. Para poder enviar / recibir tramas desde el PC.
SM	0	Configuración del modo Sleep. Al ser coordinador no entra en modo Sleep.
SP	0x300	Periodo de Sleep de 7s. Se indica un periodo mayor o igual que el mayor de la red. Pasados 2,5*SP los paquetes indirectos no enviados se descartan.
ST	0x2710	Tiempo sin actividad antes de ir a Sleep. (10s). Se configura igual que en actuador. Para que cuando este activo pase a comunicaciones directas. Como se le envían tramas cada 5s, no entra en sleep el actuador hasta que la salida de frecuencia=0.

Tabla 5.3: Configuración del módulo base

5.3.2 Descripción del GUI

En este apartado se describe la parte software encargada del control de la aplicación. Dispone de controles manuales que puede manejar el operador, así como de un modo automático, en el quela aplicación intenta mantener la consigna indicada por el operador mediante su algoritmo de control.

Para la visualización de los niveles dispone de la interfaz mostrada en la Ilustración 5.13, que será la encargada de mostrar el nivel del tanque, así como de proporcionar los mandos para poder manipular la bomba, tanto en modo manual, como en modo automático. Toda esta comunicación se realiza a través del puerto serie hacia el XBee base. Además, esta aplicación guarda en una base de datos los registros de todas las tramas recibidas y enviadas.

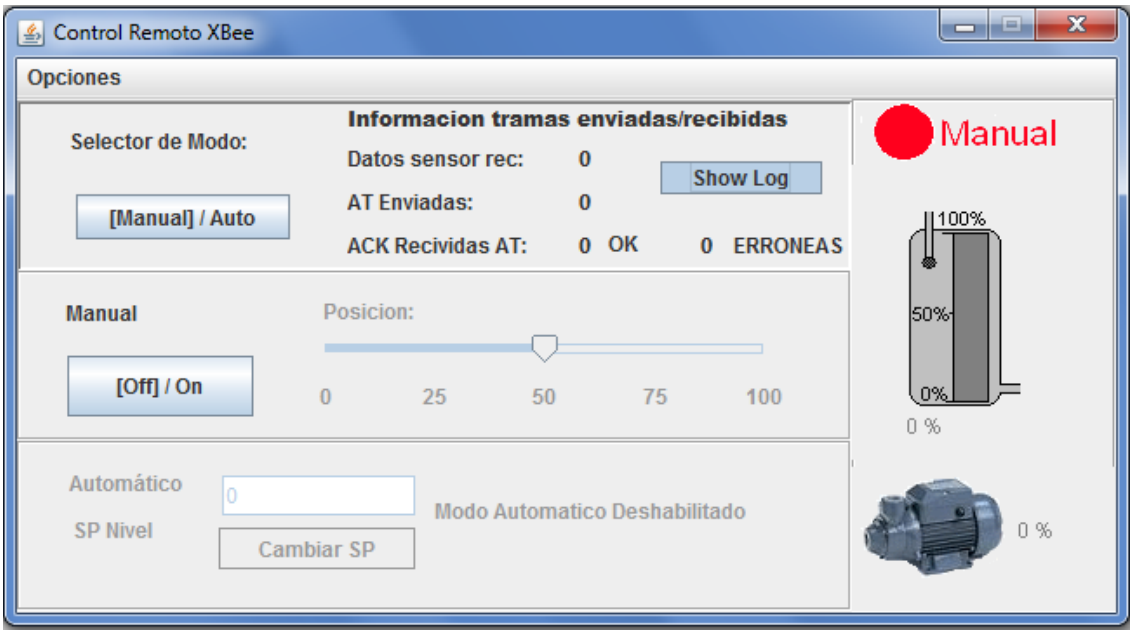


Ilustración 5.13: Pantalla de Visualización y control de la aplicación

En la aplicación se visualiza el nivel del tanque mediante su valor numérico en tanto por ciento y mediante su representación en el sinóptico. También se visualiza la cantidad de salida de control que se envía a la bomba, como porcentaje de la máxima velocidad que tiene configurado el variador que controla la bomba.

Con el modo manual activado, si se actúa sobre la barra de desplazamiento de la parte central, se producen diferentes salidas hacia la bomba según se desplace la barra. Por ejemplo, si se sitúa la barra en el 75% de su recorrido, tal y como se muestra en la Ilustración 5.14, se obtendrá en la salida del XBee una salida de 2,47 V. Haciendo cálculos se observa a continuación que efectivamente es la salida esperada.

3,3 * 0.75 = 2,475 V (teóricos), que se corresponde con la medida tomada, 2,47 V.

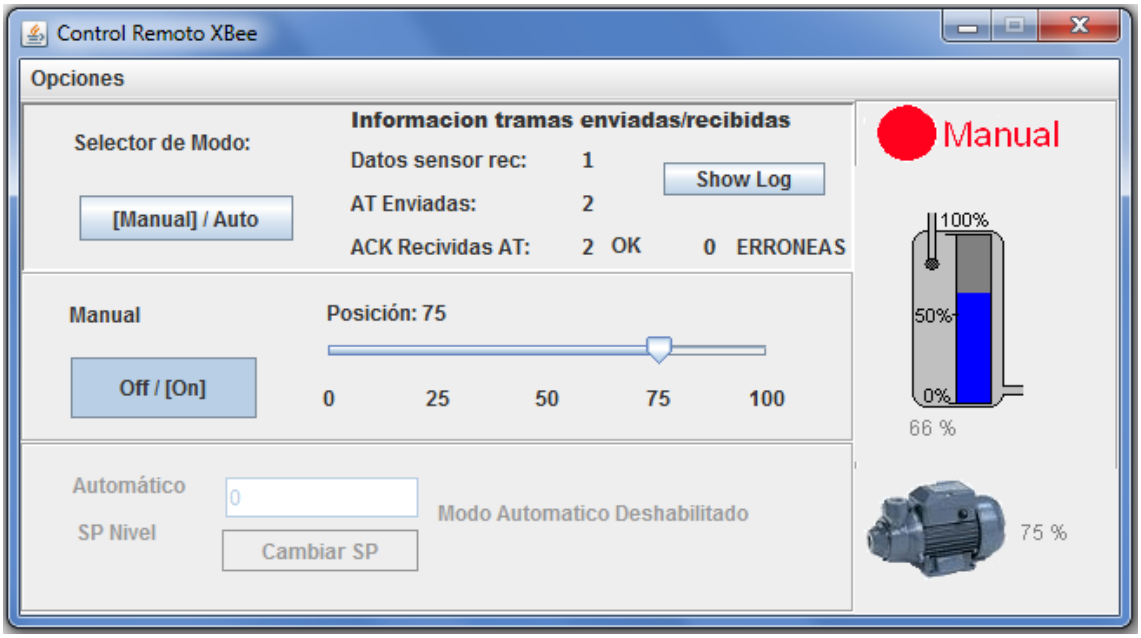


Ilustración 5.14: Utilización de la aplicación en modo manual

El nivel de llenado del tanque está siendo capturado desde el módulo sensor y actualizado cada 5 s.

Para ver las tramas que se están enviando y recibiendo de los XBee, se pulsa sobre el botón de "Show Log". Esto hace que aparezca una ventana de Log como la mostrada en la Ilustración 5.15, donde quedan reflejados todos los eventos y las diferentes conversiones producidas.

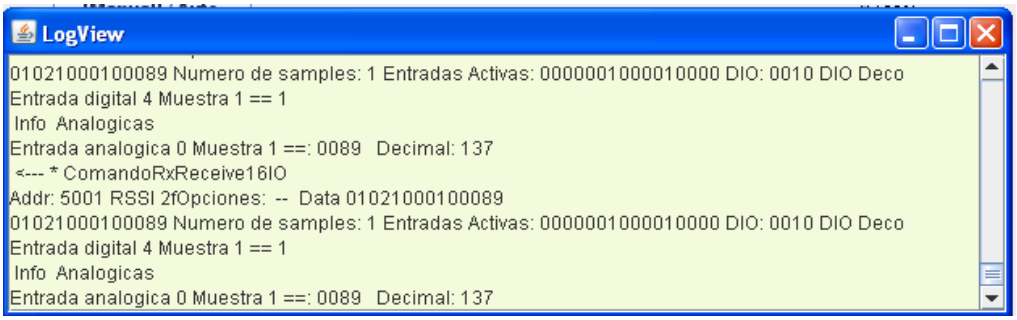


Ilustración 5.15: Log de la aplicación

Teniendo el programa en modo automático y un Set Point fijado para el tanque como se observa en la Ilustración 5.16, el programa automáticamente envía hacia la bomba la referencia adecuada para mantener el nivel consignado. Esta referencia es calculada por el algoritmo de control implementado en el programa, que se trata de un control PID digital [9]. Cuando la referencia es cero pasados 10 s, se cancela el envío de paquetes para permitir que el XBee actuador pase a estado de bajo consumo.

También implementado se ha creado un control básico On/Off. El programa, mediante su jerarquía de clases, permite fácilmente el intercambio del algoritmo de control,

permitiendo así poder disponer de diferentes algoritmos de control para el problema a controlar.

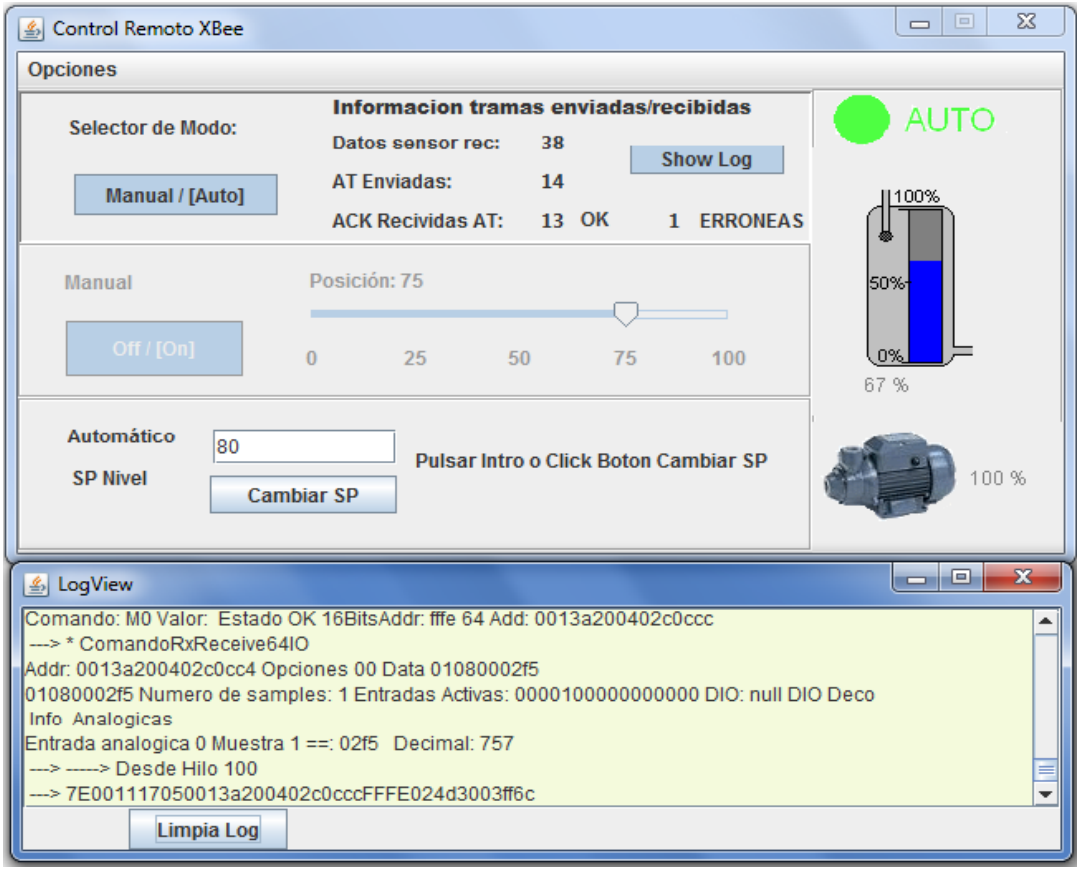


Ilustración 5.16: Funcionamiento en modo automático

Además del Log mostrado en la parte inferior de la Ilustración 5.16 y la información del número de tramas totales recibidas, enviadas y confirmaciones de recepción (*ack*) por parte del módulo actuador, que se encuentra en la parte superior de la aplicación mostrada en la Ilustración 5.16, todas las tramas que son enviadas al módulo actuador, y las que son recibidas del módulo sensor, son almacenadas en una base de datos, corriendo bajo el motor de MYSQL[10]. En la Ilustración 5.17 se observa una imagen de la base de datos con su estructura y las diversas tablas que la forman, estas son: Enviados, Recibidos y Módulos.

En la tabla enviados se tienen los siguientes campos:

- Módulo_id: Identificador del módulo al que se envía el paquete.
- Idenviado: Identificador único y secuencial del registro de la tabla de enviados.
- Fecha: Se almacena la fecha con el día, mes y año cuando el paquete ha sido enviado.

- Hora: Se almacena la fecha con precisión de segundos de cuando el paquete ha sido enviado
- Strcmd: Se almacena la trama completa enviada a los módulos.

En la tabla de recibidos se tienes los siguientes campos:

- Módulo_id: Identificador del módulo al que se envía el paquete, hace referencia a un registro de módulo dentro de la tabla Módulos.
- Idrecibido: Identificador único y secuencial del registro de la tabla de recibidos.
- Fecha: Se almacena la fecha con el día, mes y año cuando el paquete ha sido recibido.
- Hora: Se almacena la fecha con precisión de segundos de cuando el paquete ha sido recibido
- Dato: Se almacena el valor del nivel del tanque enviado por el módulo sensor.
- Strcmd: Se almacena la trama completa recibida por la estación base y transmitida al PC.

En la tabla de módulos se tienen los siguientes campos:

- módulo_id: Identificador del módulo, que lo relaciona con las otras tablas.
- Descripción: Descripción de la función de ese módulo para su identificación.
- Ubicación: Ubicación del módulo.
- Dirección: Dirección corta de red asignada a ese módulo.

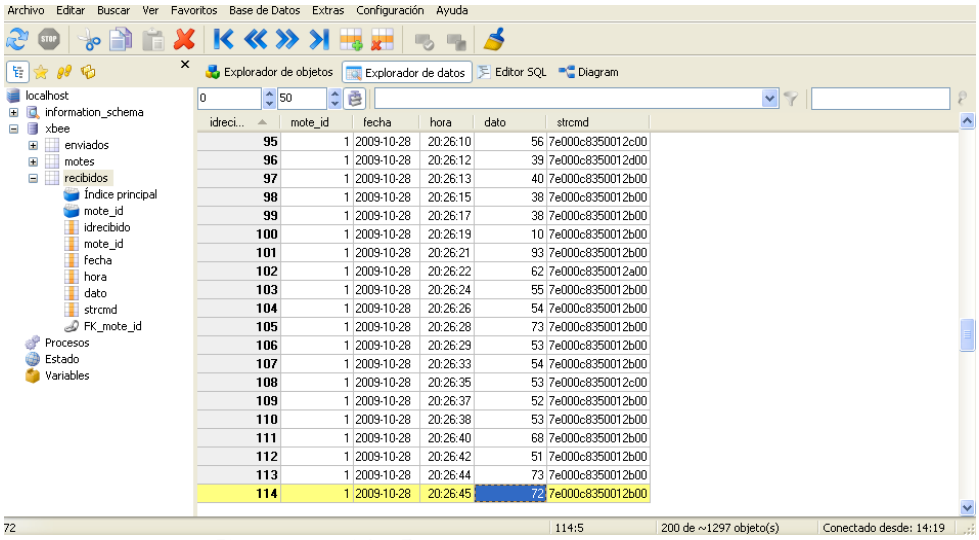


Ilustración 5.17: Base de datos de transmisiones

5.3.3 MVC

El diseño de la aplicación ha sido concebido para cumplir con el patrón de desarrollo de software MVC (*Model View Controller*, Modelo Vista Controlador)[11], para de esta manera tener independizada la parte de control, de la lógica de la aplicación y del diseño del GUI (*Graphic User Interface*, Interface gráfica de usuario), que se muestra al usuario. Con esta implementación se consigue un mayor índice de reaprovechamiento de código para aplicaciones futuras. Así, por ejemplo, si se cambia a cualquier otra aplicación de lectura de un sensor y un actuador, únicamente se debería cambiar la vista y el controlador, dejando la lógica de control, que es la más extensa sin modificar.

Cada una de las partes de este modelo tiene una funcionalidad y se interrelacionan como se muestra en la Ilustración 5.18.

Modelo: El modelo son las clases encargadas de la gestión de los datos y del comportamiento general de la aplicación. Esta parte se encarga de responder las peticiones normalmente provenientes de la vista y de responder a los cambios de estado, normalmente provocados por el controlador. De este modo, se mantiene la vista siempre actualizada con cualquier cambio ocurrido.

Vista: Es la encargada de representar el modelo y permitir la visualización de las variables, así como de canalizar la interacción del operador sobre el modelo. Es el comúnmente llamado interfaz de usuario.

Controlador: Es el encargado de gestionar las peticiones del usuario que recibe a través de la vista y pasarlas en modo adecuado al modelo, para que este envíe la respuesta directamente a la vista. También el modelo controla el interfaz que se visualiza en la vista. Por ejemplo, es el encargado de habilitar y deshabilitar botones de la vista en función de la función seleccionada.

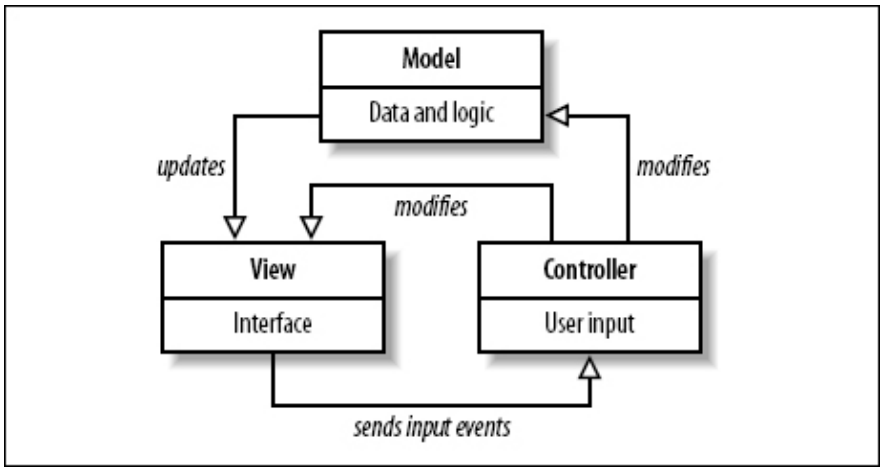


Ilustración 5.18: Diagrama del patrón MVC.

A continuación se describe brevemente la aplicación realizada y sus componentes mediante un conjunto de diagramas[12].

Diagrama de clases

El diagrama de clases permite una mejor organización del conocimiento del dominio del problema en un conjunto de abstracciones ordenadas, de forma que se obtiene un conocimiento más profundo del problema. Cada clase es la descripción de un conjunto de objetos que comparten los mismos atributos, operaciones y semántica.

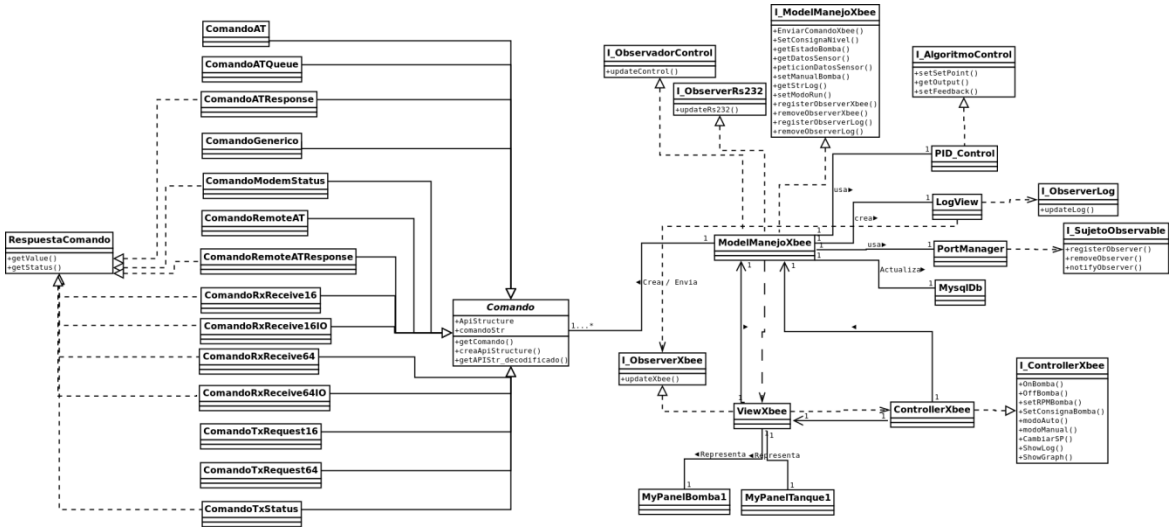


Ilustración 5.19: Diagrama de clases de la aplicación

En la Ilustración 5.19 se muestra el diagrama de clases de la aplicación, donde se observan las distintas relaciones entre las principales clases de la aplicación, así como sus principales métodos y parámetros de cada una de ellas. Se observa que casi todas las clases principales o heredan de otra clase padre como es el caso de los diferentes comandos, o implementan uno o más interfaces. Esto es así para facilitar la futura ampliación/ modificación de la aplicación. Por ejemplo, si se quiere diseñar un interfaz de usuario diferente solamente habría que modificar dos clases, la de la vista y la del controlador, implementando sus respectivos interfaces.

Diagrama de estados

Los diagramas de estados representan autómatas de estados finitos, desde el punto de vista de los estados y las transiciones que permiten expresar concurrencia, sincronización y jerarquía de objetos. Son útiles para los objetos con un comportamiento significativo. El resto se puede considerar que tienen un estado único. El estado en que se encuentra un objeto determina su comportamiento. El diagrama de estados de la aplicación principal es el mostrado en la Ilustración 5.20.

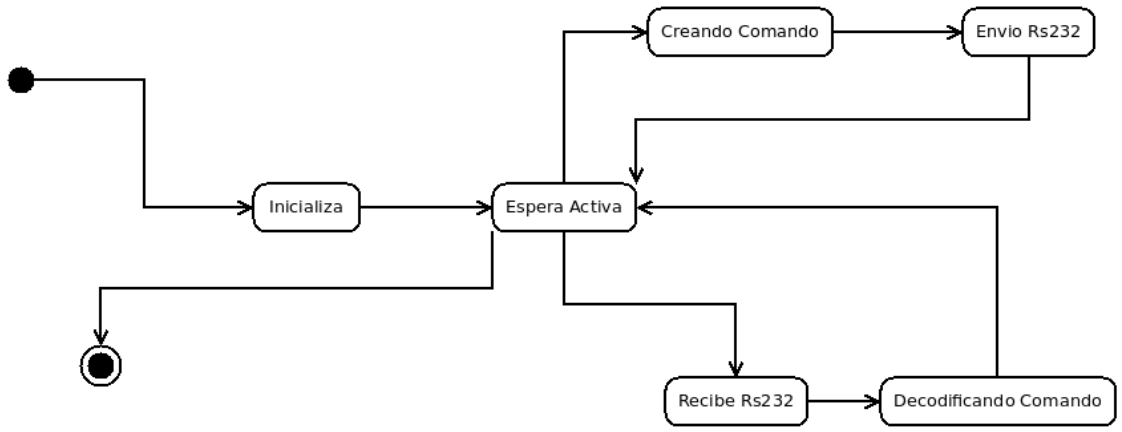


Ilustración 5.20: Diagrama de estados

En este diagrama de estados se muestran las distintas fases por donde puede pasar el XBee base, que se puede encontrar en espera activa, creando un comando para su envío o recibiendo un comando a través de la radio.

Diagrama de actividades

El diagrama de actividad es una especialización del diagrama de estado, organizado respecto de las acciones y usado para especificar, un método, un caso de uso o un proceso. Las actividades se enlazan por transiciones automáticas. Cuando una actividad termina se desencadena el paso a la siguiente actividad. Las actividades no poseen transiciones internas ni transiciones desencadenadas por eventos. El diagrama de actividades del proceso de recibir una trama es mostrado en la Ilustración 5.21-a y el de enviar una trama como respuesta a la modificación del Set Point por parte del operador es el mostrado en la Ilustración 5.21–b.

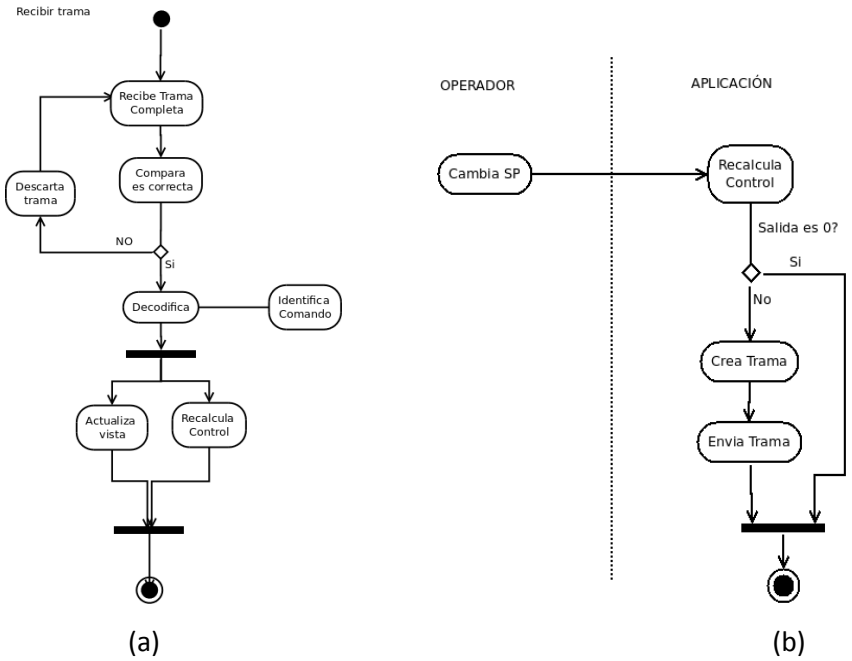


Ilustración 5.21: Diagrama de actividades a) Recibir b) Respuesta cambio SP

Diagrama de secuencia

Enmarcados dentro de los diagramas de interacción, los diagramas de secuencia resaltan la ordenación temporal de los mensajes. Son adecuados para observar la perspectiva cronológica de las interacciones. Estos muestran la secuencia de mensajes entre objetos durante un escenario concreto. En la Ilustración 5.22 se muestra el diagrama de secuencia ante un cambio del Set Point por parte del operador.

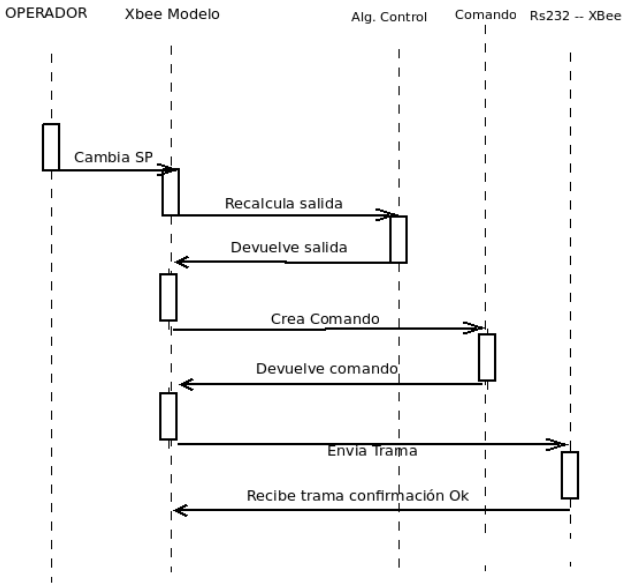


Ilustración 5.22 : Diagrama de secuencia

Diagrama de casos de uso

En el diagrama de casos de uso mostrado en la Ilustración 5.23, se aprecian las partes funcionales del programa, es decir aquellas que desarrollan una acción y como están relacionadas entre sí. También se muestra la interacción del usuario, las acciones que desencadenan y la respuesta del programa ante una información proveniente de los módulos XBee.

En este sistema se puede apreciar como el programa tiene dos formas de interacción, que son por un lado el propio usuario que puede cambiar la consigna de la aplicación o el modo de funcionamiento y el otro punto de interacción a través de la información recibida de los módulos XBee.

La recepción de información desde los módulos XBee desencadena las acciones de, asegurar la completa y correcta recepción de la trama, decodificar la información y actualizar la información en las vista. Debido a los nuevos datos recibidos si el sistema está en modo automático se recalculará el algoritmo de control, la salida se codificará en una trama con el formato definido en el API del XBee y se enviará hacia el módulo XBee correspondiente.

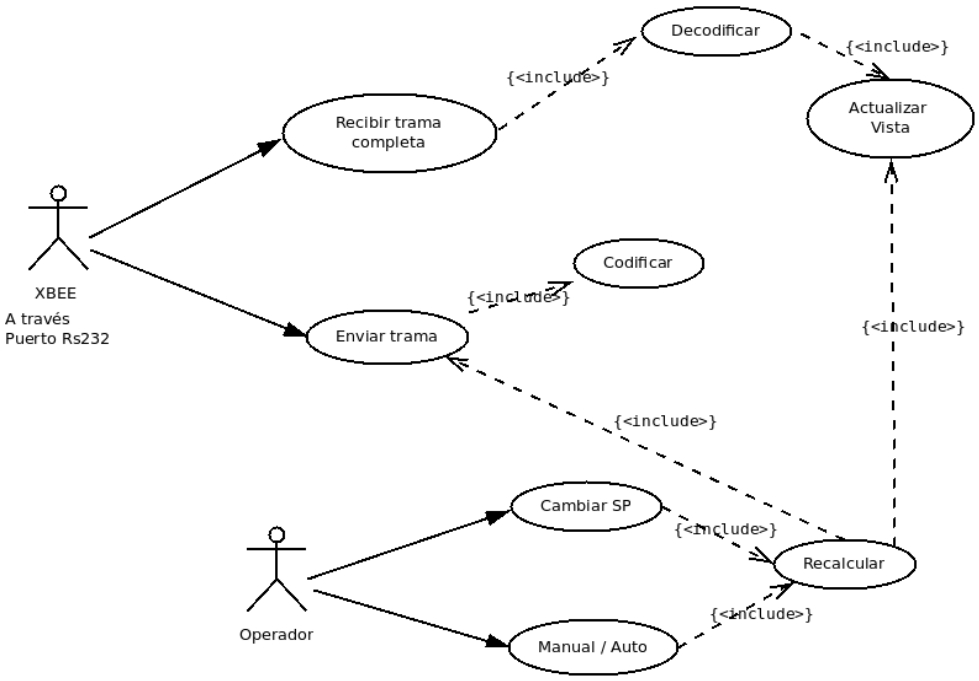


Ilustración 5.23: Diagrama casos de uso

El cronograma del funcionamiento de la aplicación, en modo manual y en modo automático, se muestra en la Ilustración 5.24:

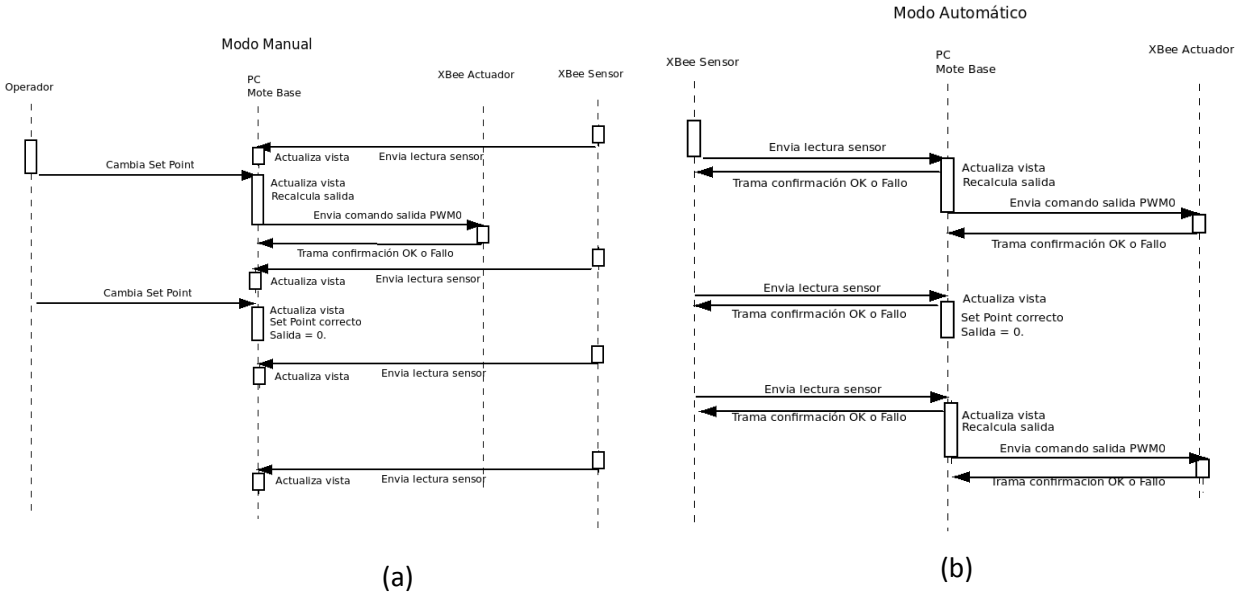


Ilustración 5.24: Cronograma funcionamiento (a)- Modo manual y (b)- Modo automático

5.4 Pruebas Realizadas

En esta sección se describen las pruebas que se han realizado para validar el correcto funcionamiento de las arquitecturas hardware y software implementadas.

Prueba 1: En modo manual se actúa desde la aplicación para producir diferentes salidas en el XBee actuador.

Prueba 2: Se conecta el XBee sensor y se varía el potenciómetro del sensor que tiene conectado para variar el nivel del tanque en la aplicación.

Prueba 3: Todo el sistema funcionando. En automático se variara el SP, y luego se variará el nivel del tanque para ver cómo responde la señal hacia la bomba.

Una vez se tiene todo montado, se prueba el correcto funcionamiento del sistema completo. Cuando se reciben las primeras tramas se toma nota de ellas y se analizan.

A continuación se describe la trama que se recibe del módulo sensor con las dos muestras de la entrada analógica AD2.

Trama recibida en módulo base con 2 muestras:

7E 00 16 82 00 13 A2 00 40 52 46 75 3A 00 02 08 08 00 08 02 54 00 08 02 54 73

0x 7E : Cabecera

0x 00 16 : Tamaño de la trama 22 bytes

0x 82 : Identificador de recepción de IO con dirección de 64 bits

0x 00 13 A2 00 40 52 46 75 : Dirección de origen 0013A20040524675 Corresponde a la dirección hardware del módulo emisor.

0x 3A : RSSI

0x 00 : Opciones 0=OK

// Comienza trama de IO

0x 02 : Número de muestras (2 muestras)

0x 08 08 : Mascara de canales activos: 0000 1000 0001 0000 : Canales activos => A2 y D3

0x 00 08 : Estado de las entradas digitales muestra 1, D3 en estado activo.

0x 0254 : Valor de la entrada analógica nº 2 muestra1. (254 / 3FF) * 3,3 V = 1,9 V

0x 00 08 : Estado de las entradas digitales muestra 2, D3 en estado activo.

0x 0254 : Valor de la entrada analógica nº 2 muestra 2 (254 / 3FF) * 3,3 V = 1,9 V

// Fin de datos IO

0x 73 : Valor del Checksum

A continuación se muestra una de las tramas enviadas al módulo actuador para fijar el valor de su salida del PWM0 al 10% de su valor máximo es decir a 0,33V.

7E 00 10 17 05 0013A200402C0CCC FF FE 02 4d 30 66 08

0x 7E : Cabecera

0x 00 10 : Tamaño de la trama 16 bytes

0x 17 : Identificador de petición de comando remoto.

0x 05 : Identificador del Frame: 05

0x 00 13 A2 00 40 2C 0C CC : Dirección de 64 bits del módulo destino. Al estar asociado esta dirección debe ser usada en vez de la de 16, ya que esta cambia a 0xFFFE

0x FF FE : Dirección corta de destino, al asociarse cambia de 0x5001 a 0xFFFE.

0x 02 : Opciones de comando, 2= Aplicar cambios inmediatamente.

0x 4D 30 66 : Comando remoto a ejecutar, M0 -> Fijar salida PWM0 a 102 (10% de 1024).

0x 08 : Checksum

5.5 Conclusiones

Se ha desarrollado un pequeño sistema de bajo coste, que ha permitido de una forma rápida proveer de un sistema de control para sistemas que están físicamente distanciados y que el cableado hubiese supuesto un gran coste. Además, dado que esta implementado el algoritmo de control sobre un PC puede ser todo lo complejo que se requiera. Esta prueba realizada sobre esta empresa comprueba como la incorporación de tecnología inalámbrica es factible y que en pocos años será una realidad en el mundo industrial.

Al amparo de los resultados aquí expuestos, ya se está estudiando la posibilidad de utilizar estos módulos para la creación de una red, para la monitorización del nivel de 16 depósitos de vino. Actualmente, este nivel ya está siendo monitorizado en un PC en la sala de control de Bodega, pero dado que el sistema fue montado hace 15 años, el modelo del convertidor de la sonda de nivel ha cambiado y ya el distribuidor no puede suministrarlo. El problema del reemplazo reside en que el modelo que proponen, no soporta el tipo de red que los antiguos si soportaban y es la que actualmente está montada en las instalación, proponiendo por parte del fabricante como única alternativa la instalación de una red nueva adaptada a los nuevos equipos y conforme se vayan rompiendo ir reemplazando los convertidores por el nuevo modelo e ir añadiéndolos a la red nueva. Dada la distancia entre los depósitos y la dificultad del cableado, debido a que las sondas y los convertidores se encuentran en la parte superior de los depósitos, se está estudiando el diseño de una red con XBee en los depósitos con el convertidor roto, y de tal manera que pueda ampliarse

fácilmente con los módulos que haya que ir colocando según se vayan sustituyendo los transmisores por el modelo nuevo. El software del PC en este caso sería simplemente una pantalla conteniendo las diferentes lecturas de los módulos conectados a la red, de forma similar a como se hace en el caso de estudio para el nivel del tanque.

5.6 Referencias

[1] Configuración en Anexo I. Variador de frecuencia V1000 de Omron. Datasheet on-line en: http://industrial.omron.es/es/products/catalogue/motion_and_drives/frequency-inverters/compact_solution/v1000/default.html

[2] Manual de operación sonda de nivel E+H FMU41 manual disponible on-line en: https://portal.endress.com/wa001/dla/5000000/1421/000/00/BA237FES _04.05.pdf

[3] Doble amplificador operacional, alimentación simple. Datasheet on-line en: http://www.datasheetcatalog.org/datasheet/fairchild/LM358.pdf

[4] Diodo Zenner, 3,6 V. Datasheet on-line en: http://www.datasheetcatalog.org/-datasheet/eic/1N4739.pdf

[5] Especificaciones de las entradas analógicas del autómata Omron C200h. Manual disponible on-line en: http://www.myomron.com//downloads/1.Manuals/PLCs/-UNITS%20CS_C200/W325-E1-04+C200H-AD003-DA003+Ope-ration_Manual.pdf

[6] Regulador lineal ajustable, Datasheet on-line en: www.ti.com/lit/ds/symlink/lm317.pdf

[7] Doble amplificador operacional de potencia. Datasheet on-line en: http://www.datasheetcatalog.org/datasheets2/11/113911_1.pdf

[8] Adaptador de niveles lógicos entre puertos RS232 y UARTs con tecnología TTL 3V. Datasheet disponible on-line en:

[9] PID Digital, detalles de la implementación en Anexo II. Katusito Ogata, "Sistemas de control en tiempo discreto",2º Edición Prentice Hall,1996.

[10] Paul DuBois "My SQL, Cookbook",Second Edition, O'Really Media, 2007

[11] Eric Freeman, Elisabeth Freeman, Kathy Sierra, Bert Bates, "Head First Design Patterns", O'Really Media, 2004.

[12] Roger S Pressman, "Ingeniería del software" McGraw-Hill , 2010.

Capítulo 6

Conclusiones y Trabajos Futuros

6.1 Conclusiones

En este proyecto se ha abordado el diseño de un sistema de bajo coste, basado en redes inalámbricas de sensores con objeto de obtener un sistema de control que permita regular sistemas automáticos alejados de la sede de la instalación principal.

Se ha realizado un estudio para escoger el módulo de radio más idóneo para la aplicación. Teniendo en cuenta, el coste, su facilidad de integración y el no requerir de compontes adicionales, tales como micro-controladores, el módulo elegido fue el XBee de Digi. Concretamente, para el desarrollo de la aplicación se han usado tres módulos XBee, denominados módulo sensor, actuador y estación base, atendiendo a su rol principal.

Se han diseñado las placas encargadas de proporcionar la alimentación a los módulos XBee, así como la electrónica de acondicionamiento de señal necesaria. En el caso de la placa estación base, se ha adaptado la conexión del XBee con el puerto RS-232 del PC. La placa actuador proporciona una salida de 0-10V, que sirve para dar la referencia de velocidad de un variador de frecuencia industrial. Además, esta PCB también proporciona dos entradas para sensores industriales de tipo 4-20mA. La placa sensor proporciona dos entradas para sensores industriales con interfaz 4-20mA, ya que esta salida es la más ampliamente usada en entornos industriales. Finalmente, se ha diseñado un software de PC específico para la aplicación. Entre otros elementos, esta aplicación tiene implementado el algoritmo de control encargado de regular el sistema.

Para el desarrollo de la aplicación se han implementado en lenguaje Java las bibliotecas necesarias para poder utilizar los módulos XBee. Cabe destacar que este paquete es reutilizable por cualquier aplicación que se quiera diseñar en Java y requiera del uso de los módulos XBee. A su vez, la aplicación específica que se ha desarrollado para el sistema de control, se ha programado basándose en el patrón de diseño Modelo-Vista-Controlador. Este patrón se usa en la actualidad por multitud de programadores, ya que las aplicaciones de dispositivos Apple están basadas en dicho patrón. Concretamente, permite diseñar software estable, reutilizable, así como facilita la continuación del desarrollo por otros programadores.

Finalmente y a modo de resumen, con este estudio se ha pretendido comprobar la viabilidad de los proyectos de monitorización y control industrial basados en redes inalámbricas de sensores, en zonas donde realizar una infraestructura con el cableado tradicional sea muy costoso.

6.2 Trabajos Futuros

En este proyecto se ha desarrollado un sistema con un único actuador en un módulo XBee y con un único sensor montado en otro módulo XBee. Como trabajo futuro para mejorar el proyecto, se podrían añadir nuevos sensores y actuadores con las siguientes posibilidades, seis ADCs o siete salidas digitales, ocho entradas digitales, así como cualquier combinación de ellas, ya que son el número máximo que permiten lo módulos XBee.

Otra posible mejora es, que en la aplicación del módulo actuador, se podría medir la temperatura del motor de la bomba, para que en caso de que la temperatura fuese excesiva se detuviera automáticamente la bomba. Además, en el módulo actuador se puede incorporar el uso de una entrada digital para conocer el estado de marcha del variador o su salida de fallo, para en caso de que por cualquier motivo haya entrado en fallo o no esté en marcha, se pueda monitorizar desde la aplicación. También en este módulo sería conveniente incorporar una salida de control para el accionamiento de las máquinas aireadoras que incorporan el aire a los tanques.

En el módulo sensor se podrían monitorizar otros parámetros que afectan al proceso de depuración como son, la temperatura del contendido del tanque, la presión del tanque, que junto con la medida de nivel da una idea de la densidad del producto, así como del nivel de oxígeno contenido en el líquido del tanque. Este último parámetro es muy importante para saber cuándo el proceso de depuración se ha completado correctamente.

Un trabajo futuro a realizar sobre la aplicación de PC, sería añadirle capacidad de ser mostrada vía Internet a través de una página web, permitiendo la monitorización de la instalación de forma remota. Esto se podría implementar haciendo uso la tecnología Java

Servlet, e instalando un servidor Tomcat en el PC donde se encuentra instalada la aplicación. Esta arquitectura proporcionaría acceso a las partes de monitorización, pero no a las de actuación, con el fin de no comprometer la seguridad de las instalaciones.

Actualmente, para incorporar un nuevo elemento en la aplicación de PC, es necesario modificar el código fuente de la aplicación, incluir el nuevo elemento a monitorizar o controlar, compilar el programa y generar el nuevo ejecutable. Una posible mejora futura sería que la propia aplicación tuviera un modo edición, donde se pudiesen incorporar nuevos módulos XBee, indicando su dirección dentro de la red y el número de entradas o salidas que tiene activas, además de asociar cada una de estas a elementos que desde una librería se podrían añadir a la aplicación. Por ejemplo, si tras la instalación se decide dosificar un determinado producto a los tanques, y se necesita conocer el nivel del depósito de producto a dosificar para rellenarlo antes de que se acabe, usando el modo edición de la aplicación se seleccionaría desde la librería un tanque con variable de nivel, se la asociaría la entrada X de un nuevo módulo XBee con dirección Y, que se trata de un módulo que se ha instalado en el tanque, junto a una sonda de presión, para conocer la cantidad de producto restante en el depósito.

Anexo I

Descripción de Parámetros del Variador de Frecuencia V1000

I.1 Introducción

En este anexo se describen los parámetros del variador de frecuencia usado en este trabajo. Específicamente, el variador de frecuencia se utiliza para controlar la velocidad de la bomba, mediante las tramas recibidas por el módulo XBee con el rol de actuador. Este nodo de la red está conectado físicamente con el variador de frecuencia. En relación al modelo y fabricante, se trata del V1000 del fabricante Omron.

I.2 Parámetros del Variador de Frecuencia V1000

En la Tabla I.1 se detallan los valores que se pueden establecer en los parámetros de configuración del variador V1000. Como se ha comentado anteriormente, este dispositivo comanda la bomba que controla el caudal de entrada de agua al tanque. En este trabajo, el V1000 se ha configurado para que acepte la referencia de frecuencia en su entrada analógica A1, como una señal comprendida en el intervalo 0-10V. Además, para realizar la programación se ha supuesto que la potencia del motor conectado es de 7,5kW.

Parámetro	Descripción	Configuraciones posibles	Valor
A1-01	Selección de nivel de acceso Selecciona los parámetros a los que se puede acceder mediante el operador digital.	0: Sólo operación 1: Parámetros de usuario 2: Nivel de acceso avanzado	2
A1-02	Selección del método de control Selecciona el método de control del variador. Nota: no inicializado con A1-03.	0: Control V/f 2: Vectorial lazo abierto (OLV) 5: Vectorial lazo abierto PM (PM)	0
A1-03	Inicializar parámetros Restablece todos los parámetros a su valor por defecto (vuelve a cero tras la inicialización).	0: Sin inicialización 1110: Inicialización del usuario 2220: Inicialización a 2 hilos 3330: Inicialización a 3 hilos	2220
Selección del modo de operación			
b1-01	Selección de referencia de frecuencia	0: Operador, valores d1-□□ 1: Entrada analógica A1 o A2 2: Comunicaciones serie, RS-422/485 3: Tarjeta opcional 4: Entrada de pulsos (terminal RP)	1
b1-02	Selección del comando marcha RUN	0: Operador: teclas RUN y STOP 1: Terminales: entradas digitales 2: Comunicaciones serie:RS-422/485 3: Tarjeta opcional conectada	1
b1-03	Selección del método de parada	Selecciona el método de parada cuando se retira la señal RUN. 0: Rampa a parada 1: Marcha libre a parada 2: Freno de inyección de c.c. a parada 3: Marcha libre con temporizador	1
b1-04	Selección de operación inversa	0: Marcha inversa activada 1: Marcha inversa prohibida	1
b1-14	Selección de orden de fase Cambia el orden de fase de salida.	0: Estándar 1: Cambiar orden de fase	0
Freno de inyección de c.c.			
b2-01	Frecuencia de inicio de freno de inyección de c.c.	Establece la frecuencia a la que se inicia el freno de inyección de c.c. cuando se selecciona Rampa a parada (b1-03 = 0). Si b2-01 < E1-09, la inyección de freno de c.c. empieza en E1-09.	0
b2-02	Corriente de freno de inyección de c.c.	Configura la corriente de freno de inyección de c.c. como un porcentaje de la corriente nominal del variador. En OLV la corriente de excitación de c.c. está determinada por E2-03.	0
b2-03	Tiempo de freno de inyección de c.c./ tiempo de excitación de c.c. al arrancar	Establece el tiempo de freno de inyección de c.c. al arrancar en unidades de 0,01 segundos. Se desactiva cuando se establece en 0,00 segundos.	0,0
b2-04	Tiempo de freno de inyección de c.c. a la parada	Establece el tiempo de freno de inyección de c.c. a la parada. Se desactiva cuando se establece en 0,00 segundos.	0,0
Aceleración/deceleración			
C1-01	Tiempo de aceleración 1	Configura el tiempo de aceleración 1 de 0 Hz a la frecuencia de salida máxima.	15
C1-02	Tiempo de deceleración 1	Configura el tiempo de deceleración 2 de la frecuencia de salida máxima a 0 Hz.	5
C1-03 a C1-08	Tiempos de aceleración/ deceleración 2 a 4	Configura los tiempos de aceleración/deceleración 2 a 4 (se establecen como C1-01/02)	--
C2-01	Curva S 1	Curva S 1 al inicio de la aceleración.	0,2
C2-02	Curva S 2	Curva S al final de la aceleración.	0,2
C2-03	Curva S 3	Curva S al inicio de la deceleración	0,2
C2-04	Curva S 4	Curva S al final de la deceleración.	0,2
C3-01	Ganancia de compensación de deslizamiento	• Aumente el valor si la velocidad es menor que la referencia de frecuencia. • Disminuya el valor si la velocidad es mayor que la referencia de frecuencia.	0,0

C3-02	Tiempo de retardo de la compensación de deslizamiento	• Disminuya la configuración si la compensación de deslizamiento es demasiado lenta. • Aumente la configuración si la velocidad no es estable.	0,0
C4-01	Ganancia de compensación de par	• Aumente esta configuración si la respuesta de par es lenta. • Disminuya esta configuración si se producen oscilaciones de velocidad/par.	0
C4-02	Tiempo de retardo de la compensación de par	• Aumente esta configuración si se producen oscilaciones de velocidad/par. • Disminuya la configuración si la respuesta de par es demasiado lenta.	0,0
C6-01	Selección de régimen de trabajo alto/ normal	0: Régimen de trabajo alto (HD) Aplicaciones de par constante 1: Régimen de trabajo normal (ND) Aplicaciones de par variable	0
C6-02	Selección de frecuencia de portadora	1: 2,0 kHz) 2: 5,0 kHz 3: 8,0 kHz 4: 10,0 kHz 5: 12,5 kHz 6: 15,0 kHz 7 a A: Balanceo PWM1 a 4 F: Definido por el usuario	1
Referencias de frecuencia			
d1-01	Referencia de frecuencia 1	Configure las referencia de multivelocidad 1	50
d1-02 a d1-16	Referencia de frecuencia 2 a 16	Configure las referencias de multivelocidad 2 a 16	0
d1-17	Velocidad de la operación jog	Velocidad de la operación jog	5
Curva V/f			
E1-01	Configuración de la tensión de entrada	Tensión de entrada	400
E1-04	Frecuencia de salida máxima	Frecuencia de salida	60
E1-05	Tensión de salida máxima	Tensión de salida máxima	400
E1-06	Frecuencia base	Frecuencia base	50
E1-07	Frecuencia de salida media	Frecuencia de salida media	10
E1-08	Tensión de salida media	Tensión de salida media	--
E1-09	Frecuencia de salida mínima	Frecuencia de salida mínima	10
E1-10	Tensión de salida mínima	Tensión de salida mínima	100
E1-13	Tensión base	Tensión base	400
Datos de motor			
E2-01	Corriente nominal del motor	Corriente nominal del motor.	7.5
E2-02	Deslizamiento nominal del motor	Deslizamiento nominal del motor en hercios (Hz). Configurado automáticamente por el autotuning dinámico.	--
E2-03	Corriente en vacío del motor	Corriente de magnetización en amperios. Configurado automáticamente por el autotuning dinámico.	--
E2-04	Polos del motor	Número de polos del motor. Dato necesario para el autotuning.	4
E2-05	Resistencia línea a línea del motor	Define la resistencia fase a fase del motor en ohmios. Configurado automáticamente por el autotuning.	--
E2-06	Inductancia de fuga del motor	Define la caída de tensión debido a la inductancia de fuga del motor como un porcentaje de la tensión nominal del motor. Configurado automáticamente por el autotuning.	--
Configuración de las entradas/salidas digitales y analógicas			
H1-01	Selección de función ED S1	Selecciona la función del terminal S1.	40
H1-02 a H1-06	Selección de función ED:S2 a S6	Selecciona la función de los terminales S2 a S6.	--
H2-01	Función SD: MA/MB	Configura la función de la salida relé MA-MB-MC.	0
H2-02	Función SD: P1	Configura la función de la salida de optoacoplador P1.	E

H2-03	Función SD: P2	Configura la función de la salida de optoacoplador P2.	10
H3-01	Selección de nivel de señal A1	0: 0 a +10 V (la entrada negativa se convierte en 0) 1: 0 a +10 V (entrada bipolar)	0
H3-02	Selección de función A1	Asigna una función al terminal A1.	0
H3-03	Ganancia A1	Establece el valor de entrada en % con la entrada analógica a 10V.	100
H3-04	Bias A1	Establece el valor de entrada en % con la entrada analógica a 0V.	0
H3-09	Selección de nivel de señal A2	0: 0 a +10 V (la entrada negativa se convierte en 0) 1: 0 a +10 V (entrada bipolar) 2: 4 a 20 mA (entrada de 9 bits) 3: 0 a 20 mA	0
H3-10	Selección de función A2	Asigna una función al terminal A2.	0
H3-11	Ganancia A2	Establece el valor de entrada en % con la entrada analógica a 10V/20 mA.	100
H3-12	Bias A2	Establece el valor de entrada en % con la entrada analógica a 0V/0mA/4mA. Configuración de las salidas analógicas (SA)	0
H4-01	Selección de monitorización AM	Introducir un valor igual a los valores de monitorización U1-x	103
H4-02	Ganancia AM	Establece la tensión de salida AM igual al 100% del valor de monitorización.	100
H4-03	Bias AM	Establece la tensión de salida AM igual al 0% del valor de monitorización.	0
H6-02	Escala de entrada RP	Configura el número de pulsos (en Hz) que es igual al 100% del valor de entrada.	1440
H6-03	Ganancia de entrada de tren de pulsos	Establece el valor de entrada en % con la entrada de pulsos con la frecuencia H6-02.	100
H6-04	Bias de entrada de tren de pulsos	Establece el valor de entrada en % con la frecuencia de entrada de pulsos de 0 Hz.	0
H6-06	Selección de monitorización MP	Introducir un valor igual a los valores de monitorización U1-..	101
H6-07	Escala de monitorización MP	Configura el número de pulsos de salida cuando la monitorización es 100% (en Hz).	1440
Protección de sobrecarga del motor			
L1-01	Selección de protección de sobrecarga del motor	Establece la protección de sobrecarga del motor 0: Desactivada 1: Motor refrigerado por ventilador estándar 2: Motor refrigerado por ventilación forzada 3: Modo vectorial	1
L1-02	Tiempo de protección de sobrecarga del motor	Establece el tiempo de protección de sobrecarga del motor en minutos. Normalmente no es necesario ningún cambio.	1
Prevención de bloqueo			
L3-01	Selección de prevención de bloqueo durante aceleración	0: Desactivado: el motor se acelera con la aceleración activada y se puede bloquear con una carga demasiado pesada o un tiempo de aceleración demasiado corto. 1: Propósito general: mantiene la aceleración cuando la corriente está por encima de L3-02. 2: Inteligente: aceleración en el menor tiempo posible.	1
L3-02	Nivel de prevención de bloqueo durante aceleración	Establece el nivel de corriente para la prevención de bloqueo durante la aceleración.	150
L3-04	Selección de prevención de bloqueo durante deceleración	0: Desactivada: deceleración como está configurada. Se puede producir sobretensión OV. 1: Propósito general: la deceleración se retiene si aumenta la tensión del bus de c.c.	1

L3-05	Selección de prevención de bloqueo durante marcha	0: Desactivada: se puede producir bloqueo del motor o sobrecarga. 1: Tiempo de deceleración 1: reduce la velocidad usando C1-02.	1
L3-06	Nivel de prevención de bloqueo durante marcha	Establece el nivel de corriente en el que empieza a actuar la prevención de bloqueo durante la marcha.	150
Autotuning			
T1-01	Selección del modo de autotuning	0: Autotuning dinámico 2: Sólo resistencia terminal 3: Autotuning dinámico para ahorro de energía (V/f)	3
T1-02	Potencia nominal	Configura la potencia nominal del motor (Kw).	15
T1-03	Tensión nominal	Configura la tensión nominal del motor (V).	400
T1-04	Corriente nominal	Configura la corriente nominal del motor (A).	7.5
T1-05	Frecuencia base	Configura la frecuencia base del motor (Hz).	50
T1-06	Polos de motor	Configura el número de polos del motor.	4
T1-07	Velocidad base	Configura la velocidad base del motor (rpm).	1440
T1-11	Pérdida de entrehierro del motor	Pérdida de entrehierro para determinar el coeficiente de ahorro de energía. Si no se conoce, déjelo en el valor por defecto.	--

Tabla I.1: Parametrización del variador V1000

Anexo II

Descripción del PID Digital

II.1 Introducción

En este anexo se describe el proceso que se ha llevado a cabo para obtener la ecuación de un PID digital. Un PID es el mecanismo de control más ampliamente utilizado para regular procesos en el sector industrial. La ecuación analógica del mismo es muy conocida y puede ser implementada mediante componentes de electrónica analógica, siendo muy habitual la implementación basada en amplificadores operaciones. Sin embargo, cuando se pretende utilizar dicha estrategia de control sobre un sistema basado en un micro-procesador, es necesario obtener una versión digital del mismo.

II.2 Obtención de la Ecuación Digital de un PID

La principal característica de un controlador PID es que actúa sobre la variable a ser manipulada mediante el uso de tres acciones de control diferentes: acción proporcional, donde la acción de de control es proporcional al error (diferencia entre la referencia y la salida de la planta), acción integral, donde la acción de control es proporcional a la integral de la señal de error y la acción derivativa, donde la acción de control es proporcional a la derivada de la señal de error actuante.

En términos analógicos la ecuación matemática que define el comportamiento del PID es la siguiente, mostrada en la Ecuación II.1:

$$m(t) = K\left[e(t) + \frac{1}{T_i}\int_0^i e(t)dt + T_d\frac{de(t)}{dt}\right]$$

Ecuación II.1: Ecuación de un PID analógico

En la ecuación anterior, e(t) representa la entrada al controlador, (el error, entre la referencia y la salida del sistema), m(t) a la señal de salida del controlador, K la ganancia proporcional, T_i es el tiempo integral (también llamado tiempo de reset) y T_d es el tiempo derivativo o tiempo de adelanto.

Como el controlador que se va a utilizar sólo recibirá como entrada valores discretos de las señales de retroalimentación del sistema, se ha de implementar la versión digital del PID mostrado en la Ecuación II.1. Discretizando la ecuación anterior y expresando el resultado mediante la trasformada Z, se obtiene la forma posicional del esquema de control PID, tal y como se muestra en la Ecuación II.2 y con las constantes definidas como se indica en la Ecuación II.3.

$$G_D(z) = \frac{M(z)}{E(z)} = K_P + \frac{K_I}{1-z^{-1}} + K_D(1-z^{-1})$$

Ecuación II.2: Transformada Z del PID

Donde:

$$K_P = K - \frac{KT}{2T_i} = K - \frac{K_I}{2}K_I = \frac{KT}{T_i}K_D = \frac{KT_D}{T}$$

Ecuación II.3: Ganancia para cada una de las acciones de control

Siendo K_P, ganancia proporcional, K_I ganancia integral y K_D ganancia derivativa, y T el periodo de muestreo. K, Ti y T_d, son la ganancia proporcional, y los tiempos integral y derivativo de la ecuación del PID analógico, respectivamente.

Realizando la antitransformada Z, y obteniendo las series numéricas, se tiene que el cálculo del PID queda como se muestra en la Ecuación II.4:

$$\mathrm{M(n) = e(n)\left[K_P * \delta(n) + K_I * [1u(n)] - K_D * [\delta(n) - \delta(n-1)]\right]}$$

$$\mathrm{M(n) = e(n) * K_P * \delta(n) + e(n)K_I * [1u(n)] - e(n) * K_D * [\delta(n) - \delta(n-1)]}$$

$$\mathrm{M(kT) = e(kT) * K_P + K_I * \sum_{k=0}^{KT} e(k) - K_D * \{e(KT) - e(KT-1)\}}$$

Ecuación II.4: Función del cálculo del control PID discreto

Finalmente, en el Listado II.1 se muestra el código necesario en pseudo-lenguaje para implementar la Ecuación II.4:

```
// Variables necesarias:
double Y;                       //  Entrada de realimentación de la salida del sistema.
double SP                       // Set point o referencia del sistema.
double err                      // Error del sistema, calculado como referencia – salida.
double err_ant                  // Error que tenía el sistema en la anterior iteración.
double  KP,KD,DI                // Ganancia de cada una de las acciones del PID.
double  P,I,D                   // Variables para la acción proporcional, integral y derivativa.
double  I_ant                   // Acción integral aplicada en la iteración anterior.
double C_out                    // Salida del control PID.
While (true)                    // Bucle a calcular en cada nueva lectura de la salida del sistema.
Wait (nueva_Lectura);           // Se espera a tener una lectura de la salida del
                                // sistema, esta deberá de ser en periodos de tiempo
                                // definidos por el Tiempo de muestreo.
err = SP – Y                    // Cálculo del error del sistema respecto al Set Point.
P = err*KP                      //Cálculo de la acción proporcional.
I = I_ant+KI*err                //Cálculo de la acción integral.
D =KD*(err – err_ant)           //Cálculo de la acción derivativa.
C_out = P + I +D                // Salida del control en forma posicional se suma cada una de
                                // las acciones de control.
I_ant = I                       // Se actualizan las variables err_ant e I_ant para que
err_ant = err                   // contengan el valor de la iteración anterior.
If (c_out  > MAX_Control)  C_out = MAX_Control       // Se ajusta la salida de control
else if (c_out <0) C_out =0     // para que no supere los limites.
SEND_SALIDA = C_out.            // Se envía la salida de control.
Fin While.
```

Listado II.1: Implementación del PID de la Ecuación II.4 usando pseudo-lenguaje

Anexo III

Descripción de la Aplicación de Control

III.1 Introducción

En este anexo se describen las diferentes clases que se han implementado para obtener la aplicación de control que se ejecuta en el PC. Concretamente, las clases se han implementado en lenguaje Java.

El anexo comienza con una descripción en forma de árbol de los diferentes paquetes de los que consta la aplicación, así como de las clases que hay contenidas en cada una de ellos. A continuación se explica el código Java de cada una de las clases, detallando cuál es su función y las principales características. Finalmente, en la sección IV, se indican las bibliotecas necesarias para poder compilar el código fuente del proyecto, y que no están incluidas por defecto en el JDK.

III.2 Descripción de los Paquetes

La aplicación de control que se ha desarrollado en este proyecto está compuesta de seis paquetes, con diferentes funciones dentro de la interfaz. Tal y como se observará posteriormente, para llevar a cabo la implementación se ha seguido el paradigma modelo-

vista-controlador (MVC, *Model-View-Controller*).Los paquetes, junto con las clases que los componen se detallan a continuación.

III.2.1 Paquete XbeeModelo1

En este paquete se alojan las clases principales de la aplicación, así como los interfaces del patrón de diseño usando una implementación de tipo MVC. La clase ModelManejoXbee es la encargada de gestionar toda la comunicación con el XBee, así como de presentar toda la funcionalidad de la aplicación. La clase ViewXbee es donde se ha implementado la interfaz grafica. La clase ControllerXbee se encarga de gestionar el comportamiento de la interfaz de usuario, así como de pasar las peticiones de información desde la vista hasta el modelo. Finalmente, la clase AppMain es el punto de entada de la aplicación, y en ella se crean y enlazan todos los objetos.

- I_ControllerXbee.java
- I_ModelManejoXbee.java
- I_ObserverLog.java
- I_ObserverRS232.java
- I_ObserverXbee.java
- I_SujetoObservable.java
- AppMain.java
- ControllerXbee.java
- ModelManejoXbee.java
- ViewXbee.java
- LogView.java
- DialogOpciones.java

III.2.2 Paquete Comandos

En el paquete comandos se ha implementado la especificación de la comunicación del XBee, es decir, es donde se ha programado toda la lógica del API de comunicaciones del XBee, existiendo una clase comando por cada tipo de comando que el XBee tiene definido en su API. Todas la clases comando heredan de una clase Comando genérica, que les da funcionalidades comunes a todos los comandos, evitando así tener que definir esta funcionalidad en cada una de las subclases. En este paquete también se encuentra la clase que define el tipo de datos usado en la lecturas de información de las entradas de los XBee, así como de la clase CreaComandoRecibido, que es la que se usa cuando se recibe una trama de datos que no ha sido decodificada, y que por tanto no se sabe de qué tipo es.

- Comando.java
- ComandoAT.java
- ComandoATQueue.java

- ComandoATResponse.java
- ComandoGenerico.java
- ComandoModemStatus.java
- ComandoRemoteAT.java
- ComandoRemoteATResponse.java
- ComandoRxReceive16.java
- ComandoRxReceive64.java
- ComandoRxReceive16IO.java
- ComandoRxReceive64IO.java
- ComandoTxRequest16.java
- ComandoTxRequest64.java
- ComandoTxStatus.java
- CreaComandoRecibido.java
- IOSample.java
- RespuestaComando.java

III.2.3 Paquete algControl

Este paquete contiene la interfaz que han de cumplir todos los algoritmos de control que se diseñen para trabajar con la aplicación, así como la clase del algoritmo de control PID que se han diseñado en esta aplicación. La implementación realizada permite añadir cualquier algoritmo de control, con sólo implementar el interface sería suficiente para ser usado en la aplicación. La clase ControlThread se encarga de ejecutar los algoritmos de control en un hilo secundario, con objeto de no entorpecer la experiencia del usuario mientras se realizan los cálculos de control.

- I_AlgoritmoControl.java
- I_ObservadorControl.java
- ControlThread.java
- PID_control.java

III.2.4 Paquete Bbdd

En este paquete engloban las clases encargadas de gestionar el flujo de datos con la base de datos, en la que se almacenan todas las tramas enviadas o recibidas por la aplicación, y que son intercambiadas por los diferentes nodos sensores/actuadores.

- DatoFechaSensor.java
- MysqlDb.java

III.2.5 Paquete Gráficos

En el paquete gráficos se han agrupado los componentes de la interfaz gráfica que se han creado para esta aplicación. En este caso han sido dos, que se corresponden con los paneles de la bomba y del tanque con nivel, de esta forma se pueden crear fácilmente nuevos paneles con diferentes contendidos y usarlos luego en la vista sin la complicación de tenerlos que crear en esta.

- ExcepcionFueraDeRango.java
- MyPanelBomba1.java
- MyPanelTanque1.java

III.2.6 Paquete RS-232

En el paquete RS-232 se encuentra la clase PortManger. Esta clase tiene la función de establecer la comunicación desde el puerto serie del PC hasta la aplicación, notificando cuando hay disponibles datos en el puerto y enviando los datos a través de este hacia la red de módulos.

- PortManager.java

III.3Descripción del Código Fuente de las Clases

PAQUETE: XbeeModelo1.

Nombre clase: I_ControllerXbee.java

Descripción: Este interface define las funciones que tienen que implementar todos los modelos a usar con esta aplicación. Como en otra implementación software basada en el paradigma MVC, la clase controlador hace de intermediaria entre la clase modelo y la clase vista. La clase controlador y la clase vista son las encargadas de realizar y de presentar las modificaciones, respectivamente.

```
package xbeeModelo1;
public interface I_ControllerXbee {
// En este interface se definen todas la funciones que ha de implementar una clase controlador,
//  estas funciones son las que utilizara la vista para modificar su aspecto o requerir acciones al modelo.
void onBomba();
void offBomba();
void setRPMBomba(int RPM_bomba);
void setConsignaBomba(int consigna);
void modoAuto();
void modoManual();
void cambioSPNivel();
void cambioSPPresion();
void cambiarSP();
void showLog(boolean show);
void showGraph(boolean showG);
void showOpciones(boolean showOp); // Consigna trabaja con unidades de ingeniería 0-100.
}
```

PAQUETE: XbeeModelo1

Nombre clase: I_Model_ManejoXbee.java

Descripción: Este interface define las funciones necesarias para manejar el módulo XBee. Además, es el encargado de enviar y recibir los comandos con el XBee, así como definir el comportamiento de la aplicación. También tiene la función de avisar a la vista para actualizarlos cambios en la aplicación.

```
package xbeeModelo1;
import comandos.Comando;
public interface I_Model_ManejoXbee {

	void initialize();
	void enviarComandoXbee(Comando comand); // Función encargada de enviar
	                                        //  el comando a los XBEE
	void setConsignaNivel(int sp_nivel);    // Función llamada por el controlador
	                                        //  para cambiar el SetPoint

	int getEstadoBomba();                   // Función llamada por el controlador para actualizar
	                                        //  el estado del actuador en la vista
	int getDatoSensor();                    // Función llamada por el controlador para
	                                        // para actualizar en la vista el nivel medido del sensor
	void peticionDatosSensor();
	void setValManualBomba(int valorManualBomba);
	String getStrLog();
	void setModoRun(boolean autoOn);
	void actualizaDatosConf(String port,int dirAct,int dirSen,int dirADC,int dirPWM);
	// Función encargada de cambiar los datos de comunicación con los XBee
	// Se ha de indicar la dirección del XBee y el número de entrada/salida.
	void registerObserver(I_ObserverXbee obsv);
	void removeObserver(I_ObserverXbee obsv);
	void registerObserver(I_ObserverLog obsv);
	void removeObserver(I_ObserverLog obsv);
	// Funciones encargadas de gestionar los observadores para que se propaguen
	// correctamente los cambios a todas las clases que se registran para ser notificadas.
}
```

PAQUETE: XbeeModelo1

Nombre clase: I_ObserverLog.java

Descripción: Este interface define las funciones que ha tener cualquier implementación de un Log, o cualquier otra aplicación que quiera obtener actualización de los datos de los comandos recibidos y enviados a los módulos XBee.

```
package xbeeModelo1;

public interface I_ObserverLog {

	void updateLog(String str_accion);

	// Cualquier clase que quiere recibir notificaciones del cambio en el Log ha de  implementar este interface
	// implementando así el método Update el cual será llamado desde la clase observada una vez se haya
	//  registrado el observador.
}
```

PAQUETE: XbeeModelo1

Nombre clase: I_ObserverRs232. java

Descripción: Este interface define las funciones que ha tener cualquier clase que quiera obtener actualización de los datos recibidos por el puerto serie. Para conseguir esto es necesario implementar esta interfaz, y por tanto, el método Update.

```
package xbeeModelo1;

public interface I_ObserverRs232 {

	public void updatedRs232();

	// Cualquier clase que quiere recibir notificaciones de datos recibidos en el puerto serie ha de  implementar
	//este interface implementando asi el método Update el cual será llamado desde la clase observada una
	//  vez se haya  registrado el observador.

}
```

PAQUETE: XbeeModelo1

Nombre clase: I_ObserverXbee. java

Descripción: Este interface define las funciones que ha tener cualquier clase que quiera obtener actualización de los tramas recibidas por el XBee.

```
package xbeeModelo1;

public interface I_ObserverXbee {

	void updateXbee();

}
```

PAQUETE: XbeeModelo1

Nombre clase: I_SujetoObservable. java

Descripción: Este interface define las funciones que ha tener cualquier clase para poder comunicar los cambios a las diferentes clases observadoras.

```
package xbeeModelo1;

public interface I_SujetoObservable {

	void registerObserver(I_ObserverRs232 obs);
	void removeObserver(I_ObserverRs232 obs);
	void notifyObserver();

}
```

PAQUETE: XbeeModelo1

Nombre clase: AppMain. java

Descripción: Clase principal e inicial de la aplicación. Tiene la función de crear un modelo y asignárselo a un controlador, que será quien creará las ventanas de la interfaz gráfica de usuario (GUI, *Graphfical User Interface*).

```
package xbeeModelo1;

public class AppMain {

	/**
	 * @ Clase principal de la aplicación. Se ejecuta para poner en marcha la aplicación.
	*/
	public static void main(String[] args) {
		I_Model_ManejoXbee modelo1=new ModelManejoXbee();
			// Se crea un objeto modelo.
		I_ControllerXbee control1=new ControllerXbee(modelo1);
			// Se crea un objeto controlador y se la pasa como
			//  argumento el modelo creado anteriormente
		// La clase controlador será la encargada de crear el objeto vista, donde se encuentra el GUI.
	}

}
```

PAQUETE: XbeeModelo1

Nombre clase: ControllerXbee. java

Descripción: Clase que implementa el interface del controlador encargado de gestionar las acciones que realiza el usuario en el GUI (lo genera la vista), y las pasa como peticiones al modelo, quien generará las respuestas directamente a la vista. El controlador crea la vista y se encarga de su gestión, es decir, si se pulsa un botón que se tienen que deshabilitar parte de los controles, es el controlador quien deshabilita y habilita estos controles de la vista, y define de este modo el comportamiento de la vista.

```
package xbeeModelo1;

// Se importan las librerías necesarias para la implementación de las funciones de la clase.
import java.util.ArrayList;
import bbdd.DatoFechaSensor;
import bbdd.JDialogGrafica;
import bbdd.MysqlDb;

public class ControllerXbee implements I_ControllerXbee{

	I_Model_ManejoXbee modelo;		// Se crea un objeto local de tipo de modelo, que luego se inicializara
						// con el pasado por argumento del constructor.
	ViewXbee view;				// Se crea un objeto vista
	LogView mylogView;			// Se crea el objeto de la venta de LOG.
	JDialogGrafica jdGraph2=null;
	DialogOpciones jdOpciones=null;		// Se crea el objeto de la ventana de configuración

	public ControllerXbee(I_Model_ManejoXbee model)		// Implementación del constructor
```

```
        // Se ha dejado solo la declaración de las funciones pero se ha quitado el código de su implementación
        // para no hacer el anexo demasiado largo, el código completo se puede ver en el CD adjunto.

        public void cambiarSP()                 // Función para cambiar la consigna a conseguir en mod Auto
        public void cambioSPNivel()             // Función para cambiar la consigna de nivel.
        public void cambioSPPresion()           // Función para cambiar la consigna de presión, esto no se usa.
        public void modoAuto()                  // Función para poner la aplicación en modo AUTO, esta acción
                        // realizada por el controlador anula en la vista los controles manuales y habilita el setPoint.
        public void modoManual()                // Idem que la anteriror pero para poner el manejo manual.
        public void offBomba()                  // Función para parar el actuador.
        public void onBomba()                   // Función para activar el actuador.
        public void setRPMBomba(int RPM_bomba       // Función para fijar el nivel de salida del actuador
        public void showLog(boolean show)       // Muestra la ventana de LOG donde estan todas las tramas envidas y
                                                //  recibidas por la aplicación, es útil en el proceso de depuración
        public void showGraph(boolean showG)
        public void showOpciones(boolean showOp)  // Muestra la ventana para establecer las opciones de
                                                  // configuración de los XBee, dirección, nº entrada, nº de salida
                                                  // y el puerto serie local usado, ejemplo COM1, COM2,etc …
}
```

PAQUETE: XbeeModelo1

Nombre clase: ModelManejoXbee. java

Descripción: Se trata de la Clase Modelo que implementa el interface modelo, y es la encargada de implementar las funciones de comunicación con el controlador, así como las funciones de actualización de la vista. Esta clase es un observador de la clase puerto RS-232, desde donde recibe la información llegada al puerto serie, y es observador de la clase control, con la que recibe las actualizaciones de las acciones de control.

Esta clase es la encargada de gestionar la comunicación con el XBee, así como de gestionar los paquetes recibidos por la radio y pasarlos al control, para que calcule la nueva salida, así como una vez que se ha realizado el cálculo, enviarla por la radio hasta el módulo actuador. Por tanto, esta clase es la encargada de a partir de la información recibida del XBee Base a través del puerto serie, convertirla en un comando, así como también discriminar que tipo de comando es, y de generar los comandos con la información que se envía desde la aplicación. Esta clase también realiza la gestión con la base de datos para almacenar todas las tramas, así como los comandos enviados y recibidos.

```
package xbeeModelo1;

// Se importan las librerías java de acceso a archives que se usaran para guardar la configuración del programa.
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.util.*;
// Se importan las librerías de control de hilos, para poder diferentes tareas ejecutándose simultáneamente.
import algControl.ControlThread;
// Se importan las librerías de los interfaces y clases creados para la aplicación.
import algControl.I_AlgoritmoControl;
import algControl.I_ObservadorControl;
import algControl.PID_control;
import bbdd.*;
import comandos.*;
import rs232.PortManager;
```

```
// Se puede observar como el modelo implementa la propia interfaz de modelo para poder ser usado por el controlador y
por la vista, el interface observador RS232 lo que le permite poder registrarse y obtener notificaciones cada vez que llegan
nuevos datos al puerto serie, y la interfaz ObservadorControl, que le permite ejecutar las acciones de control, sin tener en
cuenta la implementación concreta del algoritmo de control usado.

public class ModelManejoXbee implements I_Model_ManejoXbee, I_ObserverRs232, I_ObservadorControl{

        // Se cran la variables de arrayLista que contendrán referencias a los objetos que se registren para recibir las
        // notificaciones del cambios en el modelo y cambios en el LOG respectivamente.

        ArrayList<I_ObserverXbee> observadores = new ArrayList<I_ObserverXbee>();
        ArrayList<I_ObserverLog> observerLog = new ArrayList<I_ObserverLog>();
        // Declaración del resto de variables usadas en la aplicación, se intenta que la mayor parte de estas serán
        //  declaradas de tipo privado pues no deben de ser accedidas desde fuera de la clase.
        private int datoSensor1;
        private int valorBomba;
        private int consignaNivel;
        private PortManager pm;
        private CreaComandoRecibido cCRecv;
        private String comandoRecibido;
        private String strToLog;
        private boolean modoAuto;  // Indica el modo en que está funcionando el control.
        private I_AlgoritmoControl algoritmoC;
        private MysqlDb xbeeDb;
        private int contador_actuaciones;
        String comandoAt_act=null;
        String direccionDestino_act=null;
        int numADCsensor=-1;
        String AddrOrigSens=null;
        String puertoRsInicial=null;

        private ControlThread cth;
        // Rs232 rs232d

        public ModelManejoXbee()
        public void enviarComandoXbee(Comando comand)
        public int getDatoSensor()
        public int getEstadoBomba()
        public void initialize()

        // Petición forzada de la lectura de las entradas activas del XBee, que actúa como sensor.
        public void peticionDatosSensor()

        /*
         * Sirve para desde el controlador actualizar la consigna de nivel
         * que ha cambiado en el GUI.
        */
        public void setConsignaNivel(int sp_nivel)
        public void setModoRun(boolean AutoOn )

        // Operaciones de manejo del patrón Observador.
        // Función usada para registrar nuevos objetos a ser notificados que han de ser implementar
        // el interface observable.
        public void registerObserver(I_ObserverXbee obsv) {

                observadores.add(obsv);
        }
        //  Elimina objetos de la lista, estos dejan de recibir notificaciones, esta función busca dentro del array la
        //  posición del objeto pasado como argumento y lo elimina de la lista.
        public void removeObserver(I_ObserverXbee obsv) {

                //observadores.remove((I_ObserverXbee)obsv);
                int j=observadores.indexOf(obsv);
                if(j>=0)
                {
                        observadores.remove(j);
                }
        }
```

```
	// Función encargada de recorrer todo el array de objetos observables y llamar a su función de update para
	// notificarles que se han producido cambios.
	public void notifyObserver()
	{
		for(int i=0;i<observadores.size();i++)
		{
			I_ObserverXbee observer = (I_ObserverXbee)observadores.get(i);
			observer.updateXbee();
		}
	}
	// Manejo para el observador de la ventana del Log. Su implementación completa esta en el CD.

	public void registerObserver(I_ObserverLog obsv)
	public void removeObserver(I_ObserverLog obsv)
	public void notifyObserverLog(String accion)
	// Fin manejo observadores del Log.

	public void setValManualBomba(int valorManualBomba)   // Se fuerza en manual la salida del actuador

	//Implementación del método que se ejecuta cada vez que llegan nuevos datos al puerto serie.
	// en este método se encarga de acceder al puerto serie, leer los bytes uno  a uno y formar con ellos un
	// comando, si los bytes recibidos forman un comando valido se discrimina el tipo de comando que se ha
	// recibido, si es un comando de estado de los sensores, se le extrae la información y se almacena en las
variables	// locales, a la vez cada vez que se recibe nueva información de los sensores se retroalimenta el algoritmo de
	// control para que recalcule la salida. Finalmente se llama a notifiObserver para notificar que ha habido
	// cambios provenientes de los sensores.

	public void updatedRs232() {

		if ( cCRecv.FormaComandoRecibido( pm.Read1Byte())  )
			{
				this.comandoRecibido=cCRecv.getRecComando();
				discriminaRecibido(comandoRecibido);
				System.out.println("Nivel: "+datoSensor1);
				algoritmoC.setFeedback(datoSensor1);
				// En el metodo anterior se modifica el valor del nivel.
				notifyObserver();
			}
	}

	/*
	 * Esta función envía al XBee la nueva consigna de la bomba.
	 * Este nuevo valor debe de estar entre 0 - 100 (%).
	 */
	private void comandaBomba (int valorBomba)  // En la implementación de este método se tiene en cuenta que
	//si la salida es cero, solo se envía la trama 1 vez y las siguientes no se envía con el objetivo de ahorrar energía.

	/*
	 * Este método es para inicializa el PWM del XBee, normalmente debe de haberse inicializado
	 * por medio del X-CTU con los parámetros AT necesarios.
	 */
	private void inicializaXbeePWM()
	{
		String myFrameId="01"; // Frame ID por defecto.
		String comandoAt="P0"; // Para el Adc 0
		String parametros="2"; // Aprox la mitad del fondo escala ( 3FF )
		ComandoAT coman=new ComandoAT(comandoAt,parametros,myFrameId);
		pm.WritePort(coman.getComandoByte());
		notifyObserverLog(coman.getAPIStr_decodificado());
		String myFrameId2="01"; // Frame ID por defecto.
		String comandoAt2="PT"; // Para el Adc 0
		String parametros2="0F"; // Aprox la mitad del fondo escala ( FF )
                                                                        // Esto marca la duración
		ComandoAT coman2=new ComandoAT(comandoAt2,parametros2,myFrameId2);

try {
			Thread.sleep(200); // Forzamos la espera para que no se solapen los comandos
		} catch (InterruptedException e) { e.printStackTrace(); }
		pm.WritePort(coman2.getComandoByte());
```

```
                    notifyObserverLog(coman2.getAPIStr_decodificado());
            }

private void discriminaRecibido(String str2)
{
            // Este metodo es ejecutado cuando tenemos un comando entero. Crea en primera instancia un comando
            // genérico    que utiliza para almacenar la información antes de extraerla e identificar el tipo de comando
            // concreto que es, cuando lo identifica crea un objeto de este tipo.

      ComandoGenerico ComandoRec=new ComandoGenerico(str2);
      String tipo=((ComandoGenerico)(ComandoRec)).getType();
      String descripcion="";

int i_tipo=Integer.parseInt(tipo,16);
Comando cmd=null;

switch (i_tipo)
      {

break;
case  131://case 83:
            // Es caso de ser un comando de datos de entradas/ salidas del XBee lee la información que contiene y
            // actualiza  las variables de la aplicación con la nueva lectura de los sensores.
            // El paquete de datos de IO se decodifica aparte, al encapsularlo dentro de un paquete IO.

cmd = new ComandoRxReceive16IO(str2);
            try {
if((((ComandoRxReceive16IO)cmd).getAddress()).compareTo(AddrOrigSens)==0)
{
                        IOSample ios=((ComandoRxReceive16IO)cmd).decodificaIO();
                        if(numADCsensor==-1)numADCsensor=3;
                        int datoSensorNivel=ios.getADC(numADCsensor); // 3  // Estaba 5 para el ADC0
                        //Ajusta el Adc en el rango de 0 a 100
                        datoSensor1=datoSensorNivel*100/1024;
                        insertIntoDB(datoSensor1,str2,1);  // 0 Enviar  // 1 Recibir
            }
else System.out.println("LA dirección de origen no es la correcta. Debe de ser"+AddrOrigSens);
                                    } catch (Exception e) {e.printStackTrace();}
                                    descripcion="ComandoRxReceive16IO";
      }

// Los demás case con el resto de tipos de comando,  se pueden encontrar en el CD.

            // Este método es el que va a llamar el algoritmo de control cuando haya calculado en su
            // iteración el valor de salida del control.  // Por eso el Modelo se suscribe al algoritmo.

            public void updateControl(double resultadoControl)

            // Función encargada de insertar las tramas en la base de datos.
            private void insertIntoDB(int dato,String cmd,int envio)  // 0 Enviar  // 1 Recibir
            // Función encargada de actualizar los datos de configuración de la aplicación, estos datos se almacenan en un
            // fichero de texto que se guarda en el disco duro, asi la próxima vez que inicia el programa este los lee y se
            // configura adecuadamente,

            public void actualizaDatosConf(String port,int dirAct,int dirSen,int dirADC,int dirPWM)
}
```

PAQUETE: XbeeModelo1

Nombre clase: ViewXbee. java

Descripción: Clase de la vista de la aplicación. Es la encargada de crear el GUI, es decir, las ventanas, los botones, los textos, los campos, los dibujos, etc. y configurar las características iníciales de estos controles. Implementa el interface Observador XBee para

recibir las actualizaciones de los valores de nivel del tanque y estado de la bomba, para así mostrarlas en cuanto son cambiadas por la clase Modelo. Muchas de las propiedades de ventanas y apariencia de los controles han sido diseñados mediante el diseñador gráfico "Jigloo GUI Editor", que es un plug-in del entorno de desarrollo Eclipse.

```
package xbeeModelo1;

// Se importan las librerías para la creación de la ventana de interface de usuario, la vista.

import graficos.MyPanelBomba1;
import graficos.MyPanelTanque1;
import java.awt.*;
import javax.swing.*;

// Esta clase implementa el interface ObserverXBee para asi poder suscribirse a los eventos de esta clase.

public class ViewXbee extends JFrame implements I_ObserverXbee {

	// Se crean variables locales de tipo interface Controlador y Modelo, para poder recibir las actualizaciones del
	// modelo y operar a través del controlador.
	private I_ControllerXbee controlador;
	private I_Model_ManejoXbee modelo;

	// En el constructor se inicializan las variables locales de modelo y controlador por las pasadas como
argumento		// cuando se llama a la función.
	public ViewXbee(I_ControllerXbee contro, I_Model_ManejoXbee mode ) {
		super();
		this.controlador = contro;
		this.modelo = mode;
		// Se registra para recibir las notificaciones del modelo.
		modelo.registerObserver((I_ObserverXbee)this);
	}

	// Hace visible la ventana gráfica.
	public void createView()

	// Se definen las principales características de la ventana, tales como el título, el tamaño, etc …
	private void initialize()
	// Estas funciones inicializan las diferentes zonas o paneles que componen la ventana grafica.
	private JPanel getJContentPane()
	private JPanel getJPanelDercho()
	private JPanel getJPanelIzquierdo()
	private JPanel getJPanelArribaIz()
	private JPanel getJPanelAbajoIz()
	// Estas funciones llama a una función del controlador que define el comportamiento de la aplicación al hacer
	// seleccionar cada uno de los elementos de la ventana , como botones, barras de desplazamientos, etc...
	private JToggleButton getJTButtonManualAuto()
	private JToggleButton getJTButtonOnOff()
	private JSlider getJSliderBomba()
	private JTextField getJTextFieldSPNivel()
	private JTextField getJTextFieldSPPresion()
	private JPanel getJPanelDibuTanque
	private JButton getJButtonCambiarSP()
	private JPanel getJPanelBomba()
	private JPanel getJPanelBombaDib()
	private JToggleButton getJTButtonLog()
	private JButton getJButton()
	private JMenuBar getJJMenuPrincipal()
	private JMenu getJMenuOpciones()
	private JMenuItem getJMenuItemConfig()


	// Métodos para el manejo con el Controller.
	// Estos métodos son llamados por el controlador para modificar el comportamiento de la ventana gráfica,
	// en respuesta a una acción del usuario o una actualización de datos desde el modelo.
```

```
	public void setLuzAuto(boolean AutoOn)
	public void setNivelTanque(int nuevoNivel)
	public int getValSlider()
	public void HabilitaModoAuto(boolean modo)
	public void HabilitaModoManual(boolean modo)
	public void HabilitacionSliderBomba(boolean modo)
	public void setPosicionSlider(int pos)
	public void setRefBomba(int refer)
	public String getSPNivel()
	public void setSPNivel(String sp_niv)
	public String getSPPresion()
	public void setSPPresion(String sp_pre)

	// Este método es ejecutado por el modelo cuando tiene cambios que quiere notificar, una vez que se ha
	// registrado el observador. Se encarga de actualizar los valores en la vista del sensor y del actuador.
	public void updateXbee() {

		// Este update actualiza ambos valores.
		// Se podría separar usando dos observadores.
		int nivelTanque = modelo.getDatoSensor();
		jPanelDibuTanque.setNivel(nivelTanque);
		int refBomba = modelo.getEstadoBomba();
		jPanelBombaDib.setReferencia(refBomba);
	}

}
```

PAQUETE: XbeeModelo1

Nombre clase: LogView. java

Descripción: Esta clase se encarga de generar un log con las tramas enviadas y recibidas a los XBee. Implementa los interfaces Observer_XBee y Oberser_Log, con objeto de poder ser observador de la clase Modelo y así recibir las notificaciones de las nuevas tramas enviadas y recibidas.

```
package xbeeModelo1;

public class LogView extends JFrame implements I_ObserverLog,I_ObserverXbee {
	/**
	 * constructor
	 * Para que se muestre la ventana hay que llamar al metodo showLog();
	 * Que se encarga de registrar con el modelo y crear la ventana.
	*/
	public LogView(I_Model_ManejoXbee model) {
		super();
		initialize();
		this.modelo=model;
		modelo.registerObserver((I_ObserverLog)this);
		modelo.registerObserver((I_ObserverXbee)this);
	}

	private void initialize()		// Define las características de la ventana.
	// Se inicializan los diferentes componentes de la ventana.
	private JPanel getJContentPane()
	private JScrollPane getjsp()
	private JTextArea getJTextAreaLog()
	// Función para hacer visible o no la ventana.
	public void showLog(boolean visible)
	// Funcion llamada desde el modelo cuando hay nuevas tramas de datos recibidas o enviadas para que se
	// muestren en la ventana de LOG.
	public void updateLog(String str_accion) {
```

```
                    //jTextAreaLog.setText(jTextAreaLog.getText()+"\n"+ modelo.getStrLog());
                    jTextAreaLog.append("\n ---> "+ str_accion);
                    // Esto es para posicionar el cursor al final del TextArea.
                    jTextAreaLog.setCaretPosition(jTextAreaLog.getText().length());
            }
            // Funcion llamada desde el modelo cuando hay cambios en este.
            public void updateXbee() {
                    jTextAreaLog.append("\n <--- "+ modelo.getStrLog());
                    // Esto es para posicionar el cursor al final del TextArea.
                    jTextAreaLog.setCaretPosition(jTextAreaLog.getText().length());
            }
}
```

PAQUETE: XbeeModelo1

Nombre clase: DialogOpciones. java

Descripción: Esta clase implementa un cuadro de dialogo con las opciones para poder cambiar el puerto de comunicaciones al que está conectado el XBee base, así como configurar otras opciones, tales como las direcciones de los módulos XBee sensor y actuador. Además, guarda esta información en un archivo, de modo que cuando la aplicación se vuelve a abrir se recupera la información que tenía la última vez que se abrió.

```
package xbeeModelo1;

// Se importan las librerías necesarias para la creación de la ventana y sus componentes.
import javax.swing.*;
import java.io.*;
import rs232.PortManager;

public class DialogOpciones extends JDialog {

            // Se cran la diferentes variables que representan cada uno de los elementos de la ventana, y las variables que
            // contienen la información de la configuración de los XBee.

            public String puertoSel;  //  @jve:decl-index=0:
            public int dirSensor;
            public int dirActuador;
            public int numADC;
            public int numPWM;

            public DialogOpciones(Frame owner)
            // Esta función lee los valores desde el fichero guardado en el disco duro y los define en la aplicación.
            private void actualizaValores()
            // Función para leer los valores desde un archivo de texto.
            private void readValoresFichero() {

            // Función para escribir los nuevos valores ajustados a un fichero de texto.
            private void escribeValoresFichero()

            // Funciones encargadas de inicializar y definir los componentes usados en la ventana.
            private void initialize()
            private JPanel getJContentPane()
            private JPanel getJPanelDialogCentro()
            private JComboBox getJComboBoxPuertoBase()
            private JTextField getJTextFieldDireccionSensor()
            private JTextField getJTextFieldNumIOSensor
            private JTextField getJTextFieldNumIO_Actuador()
            private JButton getJButtonAceptar()
}
```

PAQUETE: comandos

Nombre clase: Comando. java

Descripción: Se trata de una clase abstracta que define el constructor y los campos que todos los Comandos han de tener. Otras clases pueden heredar de ella con objeto de implementar sus propios métodos particulares de cada tipo de comando. En esta clase también es donde se crea el encapsulamiento RS-232, ya que es común a todos los comandos.

```
package comandos;

public abstract class Comando {

// En esta clase commando se definen cada uno de los campos que son comunes para los tipos de commando.
protected String header;
protected String MSB;
protected String LSB;
protected String checksum;
protected String ApiStructure;
protected String ATIdem;
protected String comandoStr;

public Comando()
{
header="7E";

comandoStr=header;
   }
        // Con el constructor se crea una instancia de commando a través de una cadena pasada como argumento y se
        // inicializan las campos que componen el comando.
public Comando(String comd_in)

        // Estas clases abstractas son para que sean sobrescritas por las clases que hereden de Comando.
protected abstract String creaApiStructure();
public abstract String getAPIStr_decodificado();

public String getComando()

public String toString()
        // En esta función se calcula el Checksum que se añade a las tramas, para comprobar que se reciben o envían
        //correctamente.
protected String creaChecksum()

public byte[] getComandoByte()
protected void calculaTamanos(String ApiStructure_st)
}
```

PAQUETE: comandos

Nombre clase: ComandoGenerico. java

Descripción: La clase ComandoGenerico hereda de la clase comando y sirve para tener un comando genérico del que se desconoce el tipo, en él se guardan todos los campos para después a través del método getType, obtener el tipo de comando que es y poder generar un comando de ese tipo pasándole la información que este genérico almacena.

```
package comandos;

public class ComandoGenerico extends Comando {

	// Constructor de un objeto comando genérico.
public ComandoGenerico(String comandoEntero)
	// Retorna el campo que contiene tipo de comando que es.
public String getType()
	// Retorna el campo principal donde se encuentra la información del comando recibido o enviado.
public Comando getMainComando()
	// Esta función crea los campos específicos de una trama UART y los añade a los ya existentes de una trama
	// XBee.
protected String creaApiStructure
	// Saca la información del campo de principal de la trama.
public String getAPIStr_decodificado()
	// Transforma los bytes recibidos en una cadena de texto para simplificar el manejo.
public String bytes_to_String(String bytes)
}
```

PAQUETE: comandos

Nombre clase: IOSample. java

Descripción: Clase diseñada para almacenar la información de las tramas de estado de entradas de los módulos XBee y proporcionar funciones para obtener los valores de las señales contenidas. Como argumento, se le pasa la parte de la trama de información de entradas de una trama API con información de entradas.

```
package comandos;

public class IOSample {
// Se cran las variables que contendrán la información extraída del paquete de datos IO del XBee.
int samples;
   String channelIndicator;
   String DIO;
   String DIO_decodificada;
   String AnalogIn_str;
private boolean hayDigitales;
int [] ADC;
private boolean tieneDio;
   String raw;
   String info="";
   String resul_ana="";
String resul_din="";
private String DIO_binary;

  // El constructor de la clase extrae la información a partir de una cadena de texto pasada como argumento,
  // a partir de esta cadena se examina para comprobar si contiene entradas digitales, en caso afirmativo se
  // se usan las mascaras para determinar de cuantas entradas hay lecturas y se extraen la información, también se
  // identifica el número de muestras que vienen en la trama y se extraen cada una de las entradas por cada muestra
 // lo mismo se realiza para las entradas analógicas que siempre vendrán en la trama.

public IOSample(String ioSample_str) throws Exception

// Con las siguientes funciones se consultan cada uno de los campos que componen el IOSample.
public String getDIO()
public String getChannels()
public int getSamples()
public int getADC(int numChan)
public boolean tieneDIO()
public String toString()
}
```

PAQUETE: comandos

Nombre clase: CreaComandoRecibido. java

Descripción: Esta clase se encarga de monitorizar los bytes recibidos por el puerto serie, cuando encuentra el delimitador de comienzo de una trama, va recibiendo los bytes y encolándolos para formar una trama completa.

```
package comandos;

public class CreaComandoRecibido {

// Variables usadas para la creación de un comando a partir de una secuencia de Bytes.
boolean Formando;
boolean finComando;
boolean esCrc;
int posicionByte;
int tamanoComan;
  String comanRec;

public CreaComandoRecibido()

/*
 *  Este metodo se encarga de ver si es el comienzo de una trasmisión o no, y forma una cadena que será una
 *transmisión entera.
 * Lo hace contando el número de bytes recibidos
 *  Este metodo es llamado cada vez que se recibe un byte, que es como lo manda el XBee.
 */
public boolean FormaComandoRecibido(byte pbyteRead)
 /*
  *  Antes  de llamar  a este metodo se ha de llamar a isComplete para averiguar si esta completo el comando
  *  o todavía no se ha terminado.
 */
public String getRecComando()
// Devuelve si la trama que se está recibiendo ya está completa o todavía faltan bytes.
public boolean isComplete()

  /*  Esta función tiene como entrada una array de bytes y da su representación
   *  de los valores en hexadecimal en un cadena en formato de dos caracteres por byte.
   *  Ej:  in_byte[0]=255; ==>  String_leido="FF";
   *  Ej2: in_byte[0]=5; in_byte[1]=220; ===> String leido="DC 05"
   * Cuando el número es simple 1-F, le antepone un cero en la cadena, para cumplir con el formato.
   *  La primera función tiene como entrada recibe un array y el segundo un byte solo.
   */
private static String ByteToStringHex(byte[] in_byte)
private static String ByteToStringHex(byte in_byte)
}
```

PAQUETE: comandos

Nombre clase: RespuestaComando. java

Descripción: Esta es la definición del interfaz RespuestaComando. Todos los comandos que sean una respuesta a un comando han de implementar esta interfaz, ya que define las funciones necesarias para ser un comando de respuesta.

```
package comandos;

public interface RespuestaComando {
```

```
   // Una respuesta de comando tiene los dos siguientes campos: el estado de la respuesta y el valor de esta.
public String getValue();
public String getStatus();
}
```

PAQUETE: comandos

Nombre clase: ComandoRxReceive16. java

Descripción: En esta clase se realiza la implementación de un comando de recepción de un paquete de datos genérico con dirección corta.

```
package comandos;

public class ComandoRxReceive16 extends Comando implements RespuestaComando
{

   String sourceAddr;
   String RSSI;
   String options;
   String rfData;
  //String idem="89";


public ComandoRxReceive16 (String com_res)
protected String creaApiStructure()
public String getAPIStr_decodificado()


// Implementa las funciones que le permiten ser una respuesta a un comando.
public String getValue()
public String getStatus()


public IOSample decodificaIO() throws Exception
private String srt_opciones(String opt)
}
```

PAQUETE: comandos

Nombre clase: ComandoRxReceive16IO. java

Descripción: En esta clase se implementa un comando de recepción de un paquete con dirección corta, que además incluye información del estado de las entradas/salidas de un módulo XBee.

```
package comandos;

public class ComandoRxReceive16IO extends Comando implements RespuestaComando {

   String sourceAddr;
   String RSSI;
   String options;
   String rfData;
  //String idem="89";
public ComandoRxReceive16IO (String com_res)

protected String creaApiStructure()
```

```
public String getAPIStr_decodificado()

        // Implementa las funciones que le permiten ser una respuesta a un comando.
public String getValue()
public String getStatus()

public IOSample decodificaIO() throws Exception
private String srt_opciones(String opt)
}
```

PAQUETE: comandos

Nombre clase: ComandoRxReceive64IO. java

Descripción: Esta es una implementación de un comando de recepción de un paquete con dirección larga, con información del estado de las entradas/salidas de un XBee.

```
package comandos;

public class ComandoRxReceive64IO extends Comando implements RespuestaComando {

   String sourceAddr;
   String RSSI;
   String options;
   String rfData;
  //String idem="82";

public ComandoRxReceive64IO (String com_res)
protected String creaApiStructure()
public String getAPIStr_decodificado()
public IOSample decodificaIO() throws Exception
        // Implementa las funciones que le permiten ser una respuesta a un comando.
public String getValue()
public String getStatus()
}
```

PAQUETE: comandos

Nombre clase: ComandoRxReceive64.java

Descripción: Esta es una implementación de un comando que es de recepción de un paquete de datos genérico con dirección larga.

```
package comandos;

public class ComandoRxReceive64 extends Comando implements RespuestaComando {

   String sourceAddr;
   String RSSI;
   String options;
   String rfData;
  //String idem="89";
public ComandoRxReceive64 (String com_res)
protected String creaApiStructure()
public String getAPIStr_decodificado()
        // Implementa las funciones que le permiten ser una respuesta a un comando.
```

```
public String getValue()
public String getStatus()
}
```

PAQUETE: comandos

Nombre clase: ComandoTxStatus. java

Descripción: Implementación de un comando de respuesta del estado del envío de datos. Concretamente, sirve para saber si se ha recibido el ACK del paquete enviado.

```
package comandos;

public class ComandoTxStatus extends Comando implements RespuestaComando
{

   String frameID;
   String status;
//String idem="89";

// En el constructor es donde se extrae la información y se guarda en las variables de la clase.
public ComandoTxStatus(String com_res)

protected String creaApiStructure()
public String getAPIStr_decodificado()
public String getStatus()

// Devuelve una cadena con el significado del estado.
public String getStatusMean()

}
```

PAQUETE: comandos

Nombre clase: ComandoRemoteATResponse. java

Descripción: Implementación de un comando de respuesta por el envió de un comando AT de configuración remoto. En este paquete se halla la respuesta al comando remoto que se envió al XBee remoto.

```
package comandos;

public class ComandoRemoteATResponse extends Comando implements RespuestaComando {

private String frameID;
private String b64BitsAddr;
private String b16BitsAddr;
private String ATCommand;
private String parameterValue;
private String ATc_char;
private String status;

        // En el constructor es donde se extrae la información y se guarda en las variables de la clase.
        public ComandoRemoteATResponse(String str_cmd)
```

```
        protected String creaApiStructure()
        public String getAPIStr_decodificado
        public String getStatus()
        public String getValue()
                // Devuelve una cadena con la descripción del estado.
        public String getStatusMean()

        // Método propio de este comando:
        // Devuelve una cadena con el Comando AT.
public String getATCommand()

}
```

PAQUETE: comandos

Nombre clase: ComandoATResponse. java

Descripción: Implementación de un comando de respuesta por el envió de un comando AT. En este paquete se halla la respuesta al comando AT enviado al módulo XBee conectado al puerto serie.

```
package comandos;

public class ComandoATResponse extends Comando implements RespuestaComando {

private String frameID;
private String ATCommand;
private String parameterValue;
private String ATc_char;
private String status;
        // En el constructor es donde se extrae la información y se guarda en las variables de la clase.
public ComandoATResponse(String com_res)
protected String creaApiStructure()
public String getAPIStr_decodificado()

public String getValue()
public String getStatus()
        // Implementa las funciones que le permiten ser una respuesta a un comando.
public String getStatusMean()


        // Método propio de este comando:
        // Devuelve una cadena con el Comando de respuesta AT.

public String getATCommand()
}
```

PAQUETE: comandos

Nombre clase: ComandoAT. java

Descripción: Esta clase implementa un comando de tipo AT de respuesta inmediata. Estos comandos se usan para configurar el XBee que se encuentra conectado por el puerto serie. Este comando sólo es de envío y se ejecuta cuando el módulo lo recibe. La clase permite, a partir del comando AT pasado como argumento, crear los campos necesarios para ser una trama del API para envío de comandos AT.

```
packagepackage comandos;

public class ComandoAT extends Comando
{

private String frameID;
private String ATCommand;
private String parameterValue;
private String ATc_char;

        // En el constructor es donde se extrae la información y se guarda en las variables de la clase.
public ComandoAT(String ATCm,String Param,String frameIden)
public ComandoAT(String ATCm,String FrameId)
public ComandoAT(String comandoEntero)

        // Estas funciones devuelven los diferentes campos que componen un comando AT.

public String getComandoATStr()
public String getAPIStr_decodificado()

}
```

PAQUETE: comandos

Nombre clase: ComandoATQueue.java

Descripción: Esta clase implementa la generación de comando de tipo AT de respuesta encolada, es decir, no se ejecuta hasta que el módulo recibe el comando ATAC (aplicar cambios). Este comando sólo es de envío y se encarga de, a partir del comando AT pasado como argumento, crear los campos necesarios para ser una trama del API para envío de comandos AT.

```
package comandos;
/**
 *
 * @author jose
 */
public class ComandoATQueue extends Comando
{
private String frameID;
private String ATCommand;
private String parameterValue;
private String ATc_char;

        // En el constructor es donde se extrae la información y se guarda en las variables de la clase.

public ComandoATQueue(String ATCm,String Param,String frameIden)
public ComandoATQueue(String ATCm,String FrameId)
public ComandoATQueue(String comandoEntero)

        // Estas funciones devuelven los diferentes campos que componen un comando AT Encolado.

public String getComandoATStr()
protected String creaApiStructure()
public String getAPIStr_decodificado()
}
```

PAQUETE: comandos

Nombre clase: ComandoRemoteAT. java

Descripción: Esta clase implementa la generación de comando remoto de tipo AT. Este comando se envía a través del módulo conectado al puerto serie, para que se ejecute en un módulo en el alcance de la radio de este, y devuelva un paquete remoto con la respuesta. Estos comandos se pueden usar para cambiar la configuración de los módulos actuador y sensor sin tener que ir físicamente al sitio donde están ubicados, o para forzar escaneos de las entradas, así como de alguno de los valores de las salidas.

```
package comandos;
public class ComandoRemoteAT extends Comando
{
private String frameID;
private String ATCommand;
private String parameterValue;
private String ATc_char;
private String DestinationAdd64;
        private String DestinNet16;
        private String CommandOpt;

        // En el constructor es donde se extrae la información y se guarda en las variables de la clase.

public ComandoRemoteAT(String ATCm,String Param,String frameIden,String destinationAddr)
public ComandoRemoteAT(String ATCm,String FrameId,String destinationAddr)
public ComandoRemoteAT(String comandoEntero)

        // Estas funciones devuelven los diferentes campos que componen un comando Remoto AT.

public String getComandoATStr()
protected String creaApiStructure()
public String getAPIStr_decodificado()

}
```

PAQUETE: comandos

Nombre clase: ComandoModemStatus. java

Descripción: Esta clase implementa un comando de respuesta de estado del XBee. Estos paquetes de estado los envía el XBee cuando se resetea, cuando termina la asociación, cuando empieza a ser coordinador de una red, etc..

```
package comandos;

public class ComandoModemStatus extends Comando {

private String cmdData;
        // En el constructor es donde se extrae la información y se guarda en las variables de la clase.
ComandoModemStatus(int num)
        // Estas funciones devuelven los diferentes campos que componen un comando de estado.
public ComandoModemStatus(String comandoEntero)
protected String creaApiStructure()
public String getAPIStr_decodificado()
}
```

PAQUETE: comandos

Nombre clase: ComandoTxRequest16. java

Descripción: Esta clase implementa la creación de un comando de envío genérico de datos por parte del XBee hacia otro XBee en su red, mediante la dirección corta o dirección de red.

```
package comandos;

public class ComandoTxRequest16 extends Comando {

private String frameID;
private String destAdd16;
private String options;
private String RFdata;
        // En el constructor es donde se extrae la información y se guarda en las variables de la clase.
public ComandoTxRequest16(String DestAddr16,int option,String frameIden,String DataRf)
public ComandoTxRequest16(String comandoEntero)
        // Estas funciones devuelven los diferentes campos que componen un comando Respuesta de Tx.
protected String creaApiStructure()
public  String getData()
public String getAPIStr_decodificado()
}
```

PAQUETE: comandos

Nombre clase: ComandoTxRequest64. java

Descripción: Esta clase implementa la creación de un comando de envío genérico de datos por parte del XBee hacia otro XBee en el alcance de su radio, mediante la dirección larga de 64 bits o dirección MAC.

```
package comandos;

public class ComandoTxRequest64 extends Comando {

private String frameID;
private String destAdd64;
private String options;
private String RFdata;;

        // En el constructor es donde se extrae la información y se guarda en las variables de la clase.
public ComandoTxRequest64(String DestAddr64,int option,String frameIden,String DataRf)
public ComandoTxRequest64(String comandoEntero)

        // Estas funciones devuelven los diferentes campos que componen un comando Respuesta de Tx.
protected String creaApiStructure()
public  String getData()
public String getAPIStr_decodificado()
}
```

PAQUETE: algControl

Nombre clase: I_AlgoritmoControl. java

Descripción: Este interface define las funciones que han de implementar las clases que se dediquen al cálculo del algoritmo de control. Tal y como se puede observar, esto permite tener diferentes algoritmos de control y cambiar entre ellos, ya que todos van a tener la misma interfaz, aunque el algoritmo que calcula la salida sea diferente.

```
package algControl;

public interface I_AlgoritmoControl {
	// Estas tres funciones básicas son las que tiene que implementar cualquier implementación de un algoritmo
	// de control sin importar la implementación concreta o el tipo de algoritmo.

	void setSetPoint(int sp);	// Sirve para fijar la consigna
	double getOutput();		// Devuelve la salida de control
	void setFeedback(int fb);	// Es la retroalimentación del lazo de control
}
```

PAQUETE: algControl

Nombre clase: I_ObservadorControl.java

Descripción: Esta interface define la funciones que tienen que tener las clases que vayan a observar a la clase que calcula la salida de control, es decir, aquellas que se les va actualizar el valor de la salida del controlador cuando se realice un nuevo cálculo. De este modo, no es necesario que el programa espere el cálculo del algoritmo de control, que en algunos casos debido a los cálculos complejos puede ser lento.

```
package algControl;
public interface I_ObservadorControl {
	public void updateControl(double resultadoControl);
}
```

PAQUETE: algControl

Nombre clase: ControlThread.java

Descripción: En esta clase se implementa la creación de un hilo de ejecución paralelo al programa para realizar el procesamiento del algoritmo de control. De este modo, no es necesario que el programa este esperando que se calcule la señal de control. Como se ha comentado anteriormente, el algoritmo de control se ejecuta en un hilo autónomo, así puede estar esperando que le lleguen los paquetes con la información del estado del sistema, y a partir de ésta, generar las salidas de control y volver al modo de espera, hasta la próxima recepción del siguiente dato de información de estado del sistema.

```
package algControl;

import java.util.ArrayList;
import xbeeModelo1.I_ObserverXbee;

public class ControlThread extends Thread
{
```

```
	// Variables necesarias para la gestión del hilo.
	I_AlgoritmoControl algoritmo;
	boolean activo;
	boolean suspendido;
	ArrayList <I_ObservadorControl> listaObservadores;

	// Inicializa variables y parametros necesarios
	public ControlThread(String strNombre,I_AlgoritmoControl algorit)

	public synchronized void detenerHilo()
	/*
	 * suspenderHilo(boolean estado)
	* Permite suspender la ejecución de un hilo con el Arg0 = true
	 * y la reanudación del mismo con Arg0=false;
	*/
	public synchronized void suspenderHilo(boolean estado_suspedido)

	/*
	 * Al terminar run se terminara también el hilo.
	 * Este método se ejecutara una vez se tenga creado el objeto hilo, al
	 * ejecutar el metodo myhilo.start();
	* @see java.lang.Thread#run()
	*/
	public void run()

	// Manejo de las funciones para ser suscribir borrar y notificar a sus observadores.

	public void registerObserver(I_ObservadorControl obsv)
	public void removeObserver(I_ObservadorControl obsv
	public void notifyObserver(double reuslOut)

}
```

PAQUETE: algControl

Nombre clase: PID_control implements.java

Descripción: Esta clase implementa la interfaz de algoritmo de control. Concretamente, realiza el cálculo del algoritmo de control mediante un regulador PID, para alcanzar la referencia dada(ver Anexo II para detalle de la ecuación usada).

```
package algControl;

public class PID_control implements I_AlgoritmoControl {

	int feedback;
	int referencia;
	int proporcional;
	int derivativo;
	int integral;

	int Error;				// Error = Diferencia entre salida y set point
	int ErrorLast;				// Error anterior
	int ErrorLastLast;		// Error anterior a la etapa anterior.
	double OutP;				// Salida del control PID
	boolean auto;				// Controlador en modo Auto o manual
	boolean accionDirecta;	// Acción directa o inversa del bucle
	int derivate;				// Valor de la derivada en minutos
	int reset;			// Valor de la constante integral en veces / minutos
	int setPoint;				// Set point o consigna a alcanzar.
	int Input;				// Entrada
	int InputLast;				// Entrada anterior
	double Feedback_d;
```

```
	double inte_last;			// última etapa de integración.

	// Constructor al que se le pasan las constantes del PID.
	public PID_control(int propor,int deriv,int integ)

	// Actualización de la retroalimentación
	public void setFeedback(int fb)

	public double getOutput()



	/*
	 * Implementación que mejora el comportamiento cuando el control esta en los limites
	 * y la medida cambia, pudiendo producirse que la salida repentinamente se aleje del límite.
	*/

	public double getOutput2()
	{
		setPoint=referencia;
		Input=feedback;
		derivate= derivativo;
		int Gain=proporcional;
		double OutPutTemp=0.0;
	if ( auto) {

		int InputD = Input + (Input-InputLast)*derivate*60;  // Parte derivativa
		InputLast=Input;
		Error=setPoint - InputD;
		if (!accionDirecta) Error=0-Error;
		OutPutTemp= Error * Gain+Feedback_d;  // Parte integral
		//System.out.println("Pre_Salida: "+OutPutTemp);
		if (OutPutTemp > 100 ) OutPutTemp=100;
		else if (OutPutTemp < 0) OutPutTemp=0;
		OutP=OutPutTemp;
		System.out.println("=> Post_Salida: "+OutP);
		Feedback_d=Feedback_d+(OutP - Feedback_d)*reset/60;
	 }
	else
	{
		InputLast=Input;
		Feedback_d=OutP;
	}

		return OutP;
	}

		// Función para indicar al algoritmo la consigna a conseguir.
	public void setSetPoint(int sp)

		// Función que escala la salida entre los niveles dados como máximo y mínimo.
	private double ajusta (double min,double max,double medida,double ganancia)
}
```

PAQUETE: bbdd

Nombre clase: MysqlDb.java

Descripción: Esta clase implementa la gestión con la base de datos. La clase proporciona sencillas funciones para introducir y leer las tramas, así como valores en la base de datos.

```
package package bbdd;

import java.sql.*;
import java.text.*;
import java.util.*;

public class MysqlDb {
	// Variables necesarias para la conexión con la BBDD.
	Connection conexionBD;
	String url_bd;
	String user;
	String password ;

	// En el constructor se definen los parametros de la base de datos creada para la aplicación.
	public MysqlDb()
	{
		url_bd="jdbc:mysql://localhost/xbee";
		user = "xbee_user";
		password = "xbee_user";
		inicializar();
	}

	// Establecer la conexión con la base de datos
	private boolean inicializar()

		// Funcion para escribir datos enviados en la base de datos, se detalla la sentencia SQL
	public boolean escribirEnvioDB(int moteid,int dato,String cmd)
	{String consulta="insert into enviados (mote_id,fecha,hora,dato,strcmd) values
	("+moteid+",'"+strFecha+"','"+strHora+"',"+dato+",'"+cmd+"')";  }

		// Funcion para escribir datos recibidos en la base de datos, se detalla la sentencia SQL.
	public boolean escribirRecibidoDB(int moteid,int dato,String cmd)
	{String consulta="insert into recibidos (mote_id,fecha,hora,dato,strcmd) values
	("+moteid+",'"+strFecha+"','"+strHora+"',"+dato+",'"+cmd+"')";}

		// Funcion para introducir un módulo nuevo en la base de datos, se detalla la sentencia SQL.
	public boolean escribirMoteBd()
	{String consulta="insert into motes (myaddr,descripcion) values('"+dir+"','"+desc+"');";}

	public boolean cerrar()			// Se cierra la conexión con la base de datos.
	private void MakeSelect()		// Muestras los motes almacenados en la base de datos
	{ResultSet res=s.executeQuery("Select * from motes"); }

	// Este método  saca los valores de la BBDD y las devuelve en un array de DatoFechaSensor.

	public ArrayList<DatoFechaSensor> sacaHorasDatos(String fecha1,String fecha2)

}
```

PAQUETE: bbdd

Nombre clase: DatoFechaSensor.java

Descripción: Esta clase implementa un tipo de datos llamado DatoFechaSensor, que se utiliza para comunicar los datos extraídos, o que se van a almacenar en la base de datos.

```
package bbdd;

import java.util.Calendar;
import java.util.Date;
```

```
public class DatoFechaSensor {
        // Este tipo de datos se compone de los siguientes campos básicos de: fecha, hora  y un entero que representa
        // el dato.
        private int dato;
        private Date fecha;
        private Date hora;

        private Date tiempo;
                // El constructor inicializa las variables a partir de datos simples pasados como argumento
        public DatoFechaSensor(int dat,Date fec,Date hor)
        // Se definen diferentes funciones para extraer cada uno de los datos de manera individual.
        public int getDato()
        public Date getFecha()
        public Date getHora()
        public int getMinute()
        public int getSecond()
        public int getHour()
        public int getMonth()
        public int getDay()
        public int getYear()

}
```

Nombre: Creación de la base de datos en MYSQL

Descripción: En esta parte del código se indican los comandos necesarios en lenguaje SQL para la creación de la base de datos en el motor MYSQL.

```
PackageCREATE TABLE `enviados` (
  `idenviado` int(11) NOT NULL AUTO_INCREMENT,
  `mote_id` int(11) NOT NULL,
  `fecha` date NOT NULL,
  `hora` time NOT NULL,
  `dato` int(11) DEFAULT NULL,
  `strcmd` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`idenviado`),
  KEY `mote_id` (`mote_id`),
  CONSTRAINT `FK_mote_id_env` FOREIGN KEY (`mote_id`) REFERENCES `motes` (`moteid`) ON DELETE
CASCADE ON

UPDATE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=19 DEFAULT CHARSET=latin1

CREATE TABLE `recibidos` (
  `idrecibido` int(11) NOT NULL AUTO_INCREMENT,
  `mote_id` int(11) NOT NULL,
  `fecha` date NOT NULL,
  `hora` time NOT NULL,
  `dato` int(11) DEFAULT NULL,
  `strcmd` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`idrecibido`),
  KEY `mote_id` (`mote_id`),
  CONSTRAINT `FK_mote_id` FOREIGN KEY (`mote_id`) REFERENCES `motes` (`moteid`) ON DELETE
CASCADE ON UPDATE

CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1

CREATE TABLE `motes` (
  `moteid` int(11) NOT NULL AUTO_INCREMENT,
  `myaddr` varchar(4) NOT NULL,
  `descripcion` varchar(50) DEFAULT NULL,
  PRIMARY KEY (`moteid`)
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=latin1
```

```
INSERT INTO `motes` VALUES (1,'1111','CONF 1','Sensor');
INSERT INTO `motes` VALUES (2,'2222','conf 2','Estacion Base');
INSERT INTO `motes` VALUES (3,'3333','Conf 3','Actuador');
INSERT INTO `motes` VALUES (4,'4444','Otra conf','Mote4 - Actuador');
```

PAQUETE: graficos

Nombre clase: MyPanelTanque1.java

Descripción: Clase donde se diseña el panel con el sinóptico del tanque y se crea la barra personalizada encargada de mostrar el nivel del mismo.

```
package graficos;

import java.awt.*;
import javax.swing.*;

public class MyPanelTanque1 extends JPanel {

                private static final long serialVersionUID = 1L;
                Color colorActual;
                boolean firstTime;
                int nivelTanque; // 0 – 100

                // En el constructor se inicializan las variables creadas anteriormente.
                public MyPanelTanque1()

                // Sobrescribiendo el método paintComponent, se introducen los cambios gráficos personalizados
                // al panel que estamos creando, incorporando la imagen deseada y la escala como una línea que se
                // se va dibujando de rayas horizontales según el nivel 0-100 que se le indique.
                public void paintComponent(Graphics g)

                // En esta función se repintan las zonas que han cambiado con el nuevo nivel.
                public void setNivel(int nuevoNivel)

                // Función de escalado entre el valor indicado 0-100 y los valores necesarios para dibujar el nivel.
                private int scaleTank(int nivel)

                // Permite cambiar el color de la barra gráfica.
                public void setColor(float new_col)

}
```

PAQUETE: graficos

Nombre clase: MyPanelBomba1. java

Descripción: Clase donde se diseña el panel con el sinóptico de la bomba y se crea el indicador de la velocidad en unidades porcentuales.

```
package graficos;

import java.awt.*;
import javax.swing.*;

public class MyPanelBomba1 extends JPanel {
```

```
	/**
	 * Tamaño del panel 88 X 63 pixels;
	*/
	private static final long serialVersionUID = 1L;
		private Color colorActual;
		private boolean firstTime;
		private int refBomba; // 0 - 100
		private int rpmMax; // Defecto = 000;
		private int rpmMin; // Defecto = 0;
		int x_x=83;
		int y_y=58;
		// Mediante el constructor se inicializan las variables a los valores dados desde el objeto llamante.
		public MyPanelBomba1(int max_rpm,int min_rpm,int refBombaDefecto)
		public MyPanelBomba1()
			// Permite dimensionar el Panel.
		public void setDimensions(int x,int y)
			// Se sobrescribe el método paintComponent para poder diseñar el aspecto deseado.
		public void paintComponent(Graphics g)

		// Funciones para fijar y obtener respectivamente el valor del actuador bomba en el panel.

		public void setReferencia(int refeBomba)
		public int getReferencia()
		// Permite fijar el valor de velocidad que se muestra en el panel junto al dibujo.
		public void setRpm(int rpm) throws ExcepcionFueraDeRango
		public int getRpm()
		// Funciones para fijar diferentes características del panel, como los valores max y mínimo asi como
		// el color del panel
		public void setColor(float new_col)
		public void setMaxRPM(int rpm_maxima)
		public void setMinRPM(int rpm_minima)
}
```

PAQUETE: graficos

Nombre clase: ExcepcionFueraDeRango.java

Descripción: Esta clase hereda de la clase "*exception*" e implementa una excepción de paso de argumentos fuera de rango, para ser utilizada en los paneles personalizados que se han creado.

```
package graficos;

public class ExcepcionFueraDeRango extends Exception{

	private static final long serialVersionUID = 1L;

	public ExcepcionFueraDeRango(String msg){
		super(msg);
	}

}
```

PAQUETE: rs232

Nombre clase: PortManager.java

Descripción: Esta clase permite gestionar la comunicación con el puerto serie. Específicamente, la clase proporciona métodos para abrir el puerto, cerrarlo, cambiar la configuración, mostrar los puertos disponibles, escribir por el puerto serie y leer los bytes que llegan al puerto serie. Cuando esta clase tiene datos disponibles, avisa a sus observadores por medio del método notifyobserver, los cuales al tener el aviso de que hay nuevos datos pueden ejecutar las funciones de lectura de datos que la clase proporciona.

```
package rs232;

import java.io.*;
import java.util.*;
import javax.comm.*                    // Librería para la gestión del puerto serie, no viene por defecto con el  JRE.

import xbeeModelo1.I_ObserverRs232;
import xbeeModelo1.I_SujetoObservable;

        public class PortManager implements SerialPortEventListener,I_SujetoObservable {

        // Variables para el manejo del puerto
                CommPortIdentifier IdCurrentPort;
                SerialPort serialport;
                int baudRate;
                ArrayList<I_ObserverRs232> observador232=null;

                // Constructor para inicializar la comunicación con el Puerto, indicando el Puerto y la velocidad
                public PortManager( String nom_puerto, int baudRate )
                public PortManager()

                // Función para abrir el puerto. El puerto ya no se puede usar desde otra aplicación.
                public boolean OpenPort()
                private boolean OpenPort(CommPortIdentifier cpi)

                // Función para cerrar el puerto. Es importante cerrarlo para dejarlo libre.
                public boolean ClosePort()

                // Muestra los puertos serie disponibles en el PC donde se ejecuta.
                public static boolean ShowPorts()

                // Crea un listado en un array con los puertos serie libres del PC donde se ejecuta.
                public static String[] getPorts()

                // Función utilizada para cambiar de puerto. Esta función se llama desde la ventana de
                // configuración de la aplicación principal
                public boolean ChoosePort()

                // Funciones para transmitir por el puerto serie, bien a partir de una cadena o de un array de bytes.
                public void WritePort(String str)
                public void WritePort(byte[] mybytes)

        // Funciones para la lectura del puerto serie, bien devuelve una cadena o un array de bytes.
        public String ReadPort()
        private String ReadPort(SerialPort serialPor)
        public byte Read1Byte( )
        private byte Read1Byte(SerialPort serialPor)

        /*  Este metodo tiene como  entrada una array de bytes y devuelve su representación
          *  de los valores en hexadecimal en un cadena en formato de dos caracteres por byte.
        */
        private String ByteToStringHex(byte[] in_byte)
        private String bytesToString(byte[] mybytes)

        // Función para cambiar el puerto sobre el que se trabaja.
        public void changePort(String newPort)
```

```
	// Metodo que se ejecutara cuando se lancen los eventos del puerto.
	public void serialEvent(SerialPortEvent event)

		// En esta función se notificara a las clases que estén observando esta clase.
		public void notifyObserver()
		// Métodos para manejar las subscripciones.
		public void registerObserver(I_ObserverRs232 obs
		public void removeObserver(I_ObserverRs232 obs)

	// La funcionalidad del puerto serie se ejecuta en un nuevo hilo para no ralentizar el programa principal
	// esperando la llegada de datos por el puerto.
	class ReadThread extends Thread{
		SerialPort serialP;
		public ReadThread(String str,SerialPort sp)
		public void run()
}
```

III.4 Bibliotecas Necesarias para la Compilación del Programa Java.

Nombre paquete: comm.jar

Descripción: Este paquete contiene las clases necesarias para el manejo del puerto serie. Al no ser un estándar de Java, es necesario utilizar este paquete que proporciona esta funcionalidad.

Ubicación: http://java.sun.com/products/javacomm/index.jsp

Nombre paquete: mysql-connector-java-5.1.5.jar

Descripción: Este paquete contiene las clases necesarias para la conexión de la aplicación con la base de datos MYSQL, así como los métodos y funciones para su gestión.

Ubicación: http://dev.mysql.com/doc/refman/5.0/en/connector-j.html

Anexo IV

Configuración de los Módulos XBee

IV.1 Introducción

En este anexo se detallan todos los comandos AT de configuración de los tres módulos XBee usados en el proyecto: el módulo estación base, el módulo actuador y el módulo sensor. Además, al final del documento, se ha incluido una tabla resumen con todos los comandos AT que se pueden configurar de los módulos XBee y su descripción.

Para la configuración de los módulos como se vio en el Capítulo 3 y Capítulo 4, se puede realizar desde el programa X-CTU, bien sea desde el terminal serie o desde la ventana de parametrización del XBee. También en este programa en la pestaña de parametrización global, está disponible un menú que permite almacenar la configuración que actualmente hay definida en el programa en un fichero, para después poder cargársela a un XBee cuando sea necesario.

A modo de resumen indicar que para acceder al modo comando del XBee por defecto se ha de introducir la secuencia "+++" y esperar un segundo, el XBee responde con un "OK". Para enviar un comando por el terminar serie, una vez el XBee está en modo comando se acompaña del prefijo AT. Ejemplo, AT*XX parámetro*, donde XX representa el comando y parámetro el valor a configurar ese comando. Para solo leer el valor de un comando se hace igual pero sin el campo parámetro.

IV.2 Módulo XBee Sensor

La configuración realizada sobre el módulo XBee con el rol de nodo sensor se ha almacenado en el fichero "MoteSensorNivel_v12.pro".

En la Tabla IV.1 se recogen todos los comandos AT que se han configurado en este módulo, así como los valores utilizados para configurar cada unos de los comandos. A modo de ejemplo, se puede observar que se ha configurado que se realicen las comunicaciones sobre el canal 0x0C (CH=C) y que la dirección de red o dirección corta del nodo es la 0x5001 (MY= 5001). En la tabla también se puede observar que la dirección destino de las tramas enviadas por este nodo es la 0x1111 (DL=1111).

XB24_15_4_10CD.mxi	[A]AP=1
FE	[A]PR=FF
0	[A]D8=0
231	[A]D7=0
10CD	[A]D6=0
0	[A]D5=0
[A]CH=C	[A]D4=0
[A]ID=3332	[A]D3=0
[A]DH=0	[A]D2=2
[A]DL=1111	[A]D1=0
[A]MY=5001	[A]D0=0
[A]RR=0	[A]IU=1
[A]RN=0	[A]IT=2
[A]MM=0	[A]IC=0
[A]NT=19	[A]IR=1338
[A]NO=0	[A]IA=FFFFFFFFFFFFFFFF
[A]CE=0	[A]T0=FF
[A]SC=1FFE	[A]T1=FF
[A]SD=4	[A]T2=FF
[A]A1=E	[A]T3=FF
[A]A2=0	[A]T4=FF
[A]EE=0	[A]T5=FF
[A]NI=SNSOR DATOS	[A]T6=FF
[A]PL=4	[A]T7=FF
[A]CA=2C	[A]P0=0
[A]SM=4	[A]P1=0
[A]ST=200	[A]PT=FF
[A]SP=7D0	[A]RP=28
[A]DP=3E8	[A]CT=64
[A]SO=0	[A]GT=3E8
[A]BD=3	[A]CC=2B
[A]RO=3	

Tabla IV.1: Configuración del módulo XBee con el rol de nodo sensor

IV.3 Módulo XBee Actuador

La configuración realizada sobre el módulo XBee con el rol de nodo actuador se ha guardado en el fichero "MoteActuador_6_02_2010.pro". En la Tabla IV.2 se recogen todos los comandos AT que se han configurado en este módulo, así como los valores utilizados para configurar cada unos de los comandos.

Como se puede observar en la Tabla IV.2, al igual que para el nodo sensor, se ha configurado que se realicen las comunicaciones sobre el canal 0x0C (CH=C). En este caso la dirección de red o dirección corta del nodo es la 0x5002 (MY=5002). En la tabla también se puede observar que la dirección destino de las tramas enviadas por este nodo es la 0x1111 (DL=1111), que como se observará posteriormente es la dirección de red del nodo con la función de estación base.

XB24_15_4_10CD.mxi	[A]AP=1
FE	[A]PR=FF
0	[A]D8=0
231	[A]D7=0
10CD	[A]D6=0
0	[A]D5=0
[A]CH=C	[A]D4=5
[A]ID=3332	[A]D3=0
[A]DH=0	[A]D2=5
[A]DL=1111	[A]D1=4
[A]MY=5002	[A]D0=0
[A]RR=0	[A]IU=1
[A]RN=0	[A]IT=0
[A]MM=0	[A]IC=0
[A]NT=19	[A]IR=0
[A]NO=0	[A]IA=FFFFFFFFFFFFFFFF
[A]CE=0	[A]T0=FF
[A]SC=1FFE	[A]T1=64
[A]SD=4	[A]T2=64
[A]A1=E	[A]T3=FF
[A]A2=0	[A]T4=FF
[A]EE=0	[A]T5=FF
[A]NI=ACTUADOR	[A]T6=FF
[A]PL=4	[A]T7=FF
[A]CA=2C	[A]P0=2
[A]SM=4	[A]P1=0
[A]ST=1F4	[A]PT=0
[A]SP=0x7D0	[A]RP=28
[A]DP=3E8	[A]CT=64
[A]SO=0	[A]GT=3E8
[A]BD=3	[A]CC=2B
[A]RO=3	

Tabla IV.2: Configuración del módulo XBee con la función de actuador

IV.4 Estación Base

La estación base incluye el módulo XBee que hace de puente entre la red inalámbrica (nodos sensor y actuador) y la red cableada (puerto serie del PC). La configuración realizada sobre este módulo se ha guardado en el fichero "MoteBaseRs232.pro".

En la Tabla IV.3 se recogen todos los comandos AT que se han configurado en este módulo, así como los valores utilizados para configurar cada unos de los comandos. Al igual que para los nodos anteriores, se ha configurado que se realicen las comunicaciones sobre el canal 0x0C (CH=C).

El módulo XBee de la estación base es el encargado de crear la red inalámbrica, a la que se pueden conectar posteriormente los nodos sensor y actuador. En este caso se crea la red con el identificador o PAN ID 0x3332 (ID=3332). En relación a la dirección de red o dirección corta del nodo es la 0x1111 (MY=1111), que como se puede recordar de la configuración de los nodos anteriores es la dirección de red configurada como destino en los nodos sensor y actuador. En la tabla también se puede observar que la dirección destino de las tramas enviadas por este nodo es la 0xFFFF (DL=FFFF). Este valor indica que el nodo realiza un *broadcast* al enviar datos a la red inalámbrica, es decir, que la trama de datos enviada será recibida por todos los nodos de la red.

XB24_15_4_10CD.mxi	[A]AP=1
FE	[A]PR=FF
0	[A]D8=0
231	[A]D7=0
10CD	[A]D6=0
0	[A]D5=0
[A]CH=C	[A]D4=0
[A]ID=3332	[A]D3=0
[A]DH=0	[A]D2=0
[A]DL=FFFF	[A]D1=0
[A]MY=1111	[A]D0=0
[A]RR=0	[A]IU=1
[A]RN=0	[A]IT=1
[A]MM=0	[A]IC=0
[A]NT=19	[A]IR=0
[A]NO=0	[A]IA=FFFFFFFFFFFFFFFF
[A]CE=1	[A]T0=FF
[A]SC=1FFE	[A]T1=FF
[A]SD=4	[A]T2=FF
[A]A1=0	[A]T3=FF
[A]A2=6	[A]T4=FF
[A]EE=0	[A]T5=FF
[A]NI=ESTACION BASE	[A]T6=FF
[A]PL=4	[A]T7=FF
[A]CA=2C	[A]P0=0
[A]SM=0	[A]P1=0
[A]ST=200	[A]PT=FF
[A]SP=7D0	[A]RP=28
[A]DP=3E8	[A]CT=64
[A]SO=0	[A]GT=3E8
[A]BD=3	[A]CC=2B
[A]RO=3	

Tabla IV.3: Configuración del módulo XBee con la función de Estación Base

IV.5 Comandos AT de Módulos XBee

Para finalizar este anexo se presenta una tabla resumen con los comandos AT configurables de los módulos XBee, así como una descripción con los valores que se pueden establecer en cada parámetro.

Nombre	Descripción	Rango	Valor por defecto
WR	*Write* - Escribe los valores de los parámetros en la memoria no volátil del XBee.	-	-
RE	Restore Defaults – Restaure la configuración de los parámetros a sus valores por defecto.	-	-
FR	Software Reset, Realiza un reset del módulo.	-	-
CH	Canal - Lee / Escribe el canal a usar en las transmisiones de radio.	0x0B – 0x1A	0x0C
ID	PAN ID. Lee / Escribe el identificador de la PAN.	0xFFFF	0x3332
DH	Parte alta de la dirección de destino.	0xFFFFFFFF	0
DL	Parte baja de la dirección de destino.	0xFFFFFFFF	0
MY	Dirección corta (16 bits) del módulo.	0xFFFF	0
SH	Número de serie, parte alta (Solo lectura)	0xFFFFFFFF	Pregrabado
SL	Número de serie, parte baja (Solo lectura)	0xFFFFFFFF	Pregrabado
RR	Reintentos XBee. Lee/Escribe el número máximo de intentos que el módulo intentará transmitir, en caso de no conseguirlo en el número de intentos configurado, comunicará el fallo.	0-6	0
RN	*Random Delay Slots*. Lee / Escribe el valor a usar por la capa MAC para evitar las colisiones en el algoritmo CSMA-CA. 802.15.4 – macMinBE.	0-3	0
MM	MAC Mode. Lee / escribe tipo de cabecera MAC a usar.	0 = Cabecera Digi 1= 802.15.4 no Ack 2=802.15.4 ACK 3=Digi no ACK	0
NI	Cadena ASCII Identificador de nodo.	20 caracteres ASCII	-
ND	Exploración de nodos. Realiza un barrido de la radio y te da como resultado la información de todos los nodos en su alcance de radio.	-	-
NT	Lee/ Escribe el tiempo de espera máximo de respuesta de los nodos para el proceso de exploración de nodos.	0x01-0xFC (x100ms)	0x19
NO	Habilita la-auto respuesta para el proceso de exploración de nodos.	0-1	0
DN	Nodo destino. Identifica un nodo a partir de su cadena ASCII grabada en el parámetro NI.	20 caracteres ASCII	-
CE	Habilita el módulo como coordinador.	0-1 0=Disp. Final 1=Coordinador	0
SC	Lee/ Escribe la lista de canales a tener en cuenta en los escaneos de módulos activos y de energía.	0-0xFFFF	0x1FFE
SD	Lee / Escribe la duración del escaneo de canales en la asociación como exponente de base 2. 2^(SD)	0-0F	4
SD	Lee / Escribe la duración del escaneo de canales en la asociación como exponente de base 2. 2^(SD)	0-0F	4
A1	Lee / Escribe las opciones de configuración para dispositivos finales.	0x0F	0

A2	Lee / Escribe las opciones de configuración para el coordinador.	0x0F	0
AI	Lee los errores producidos en el último intento de asociación.	0x13	0
DA	Fuerza, la terminación de una asociación.	-	-
FP	Fuerza, la entrega de mensajes indirectos que pueda tener el coordinador pendiente de envío.	-	-
AS	Escanea todos los canales definidos en el parámetro SC y devuelve la información de los módulos encontrados. Como parámetro se le pasa el tiempo máximo para esperar las respuestas por parte de los módulos.	0-6	-
ED	Escanea todos los canales definidos en el parámetro SC y devuelve la información de la energía de cada uno de ellos, como parámetro se le pasa la duración del escaneo en cada canal.	0-6	-
EE	Habilita la encriptación 128 bit AES (*Advanced Encryption Standard*- Estándar de encriptación avanzada*)*	0-1 1= Habilitado 0 = Deshabilitado	0
KY	Define la clave de encriptación para 128 bit AES.	Valor 16-byte	-
PL	Lee / Escribe el nivel de energía a usar por el módulo de radio en las transmisiones.	0-4	4
CA	Lee / Escribe el valor del parámetro CCA (*Clear Channel Assessment*), que fija el umbral superior de energía máxima en el canal para permitir una transmisión, por encima de este valor espera que se descongestione el canal.	0x24 – 0x50	0x2C (-44dBm)
SM	Lee / Escribe la configuración del modo de baja energía.(*Sleep Mode*)	0-5	0
SO	Lee / Escribe la configuración de las opciones del modo de baja energía.	0-4	0
ST	Lee / Escribe el valor del temporizador para que ha de cumplirse para pasar a modo de baja energía.(milisegundos)	1-FFFF (ms)	0x1388
SP	Lee / Escribe el ciclo periódico para pasar a modo de baja energía. (x 10 ms)	0x66B0 (10ms)	0
DP	Lee / Escribe el ciclo periódico de tiempo que el módulo pasa a baja energía si está configurado para asociarse pero no encuentra a un coordinador. (x 10 ms)	0x068B0 (10ms)	0x3E8
BD	Lee / Escribe la velocidad del puerto serie de comunicaciones.	0-7	3
RO	Lee / Escribe el número de caracteres que se recibirán antes de enviar el paquete por la radio.	0-0xFF	3
AP	Habilita o deshabilita el modo API.	0-2	0
NB	Lee / Escribe la paridad a usar en el puerto serie.	0-4	0
PR	Lee / Escribe la configuración de las resistencias de pull-up de las entradas, mediante un byte de estado.	0xFF Bit=1 Habilitada Bit=0 No Habilitada	0xFF
D8	Lee / Escribe las opciones de configuración de la entrada/salida 8, pin 9.	0-1	0
D7	Lee / Escribe las opciones de configuración de la entrada/salida 7, pin 12.	0-7	1
D6	Lee / Escribe las opciones de configuración de la entrada/salida 6, pin 16.	0-5	0
D5	Lee / Escribe las opciones de configuración de la entrada/salida 5, pin 15.	0-5	1
D0-D4	Lee / Escribe las opciones de configuración de la entrada/salida 7, pin 12.	0,2,3,4,5	0
IU	Habilita / Deshabilita, enviar todo los paquetes recibidos de	0-1	1

	entradas/salida a través del puerto serie.		
IT	Lee / Escribe el número de muestras de las entradas antes de ser enviados por la radio	10xFF	1
IS	Fuerza una lectura de las entradas habilitadas y devuelve el resultado a través del puerto serie. Establece el valor de las salidas digitales configuradas con el byte pasado como parámetro.	8-bit bitmat. (Cada bit representa el nivel de una salida así configurada)	-
IO	Fija el valor de los pines configurados como salidas digitales.	0-1	-
IC	Lee / Escribe el *bitmap* para habilitar la detección de cambio de nivel en las entradas configuradas.	0xFF	0
IR	Lee / Escribe el valor de la frecuencia de muestreo en ms.	0xFFFF (ms)	0
IA	Lee / Escribe la dirección de la que se aceptan paquetes *I/O passing* para cambiar el estado de los pines configurados como salidas.	0-0xFFFFFFFF FFFFFFFF	0xFFFFFFFF FFFFFFFF
T0-T7	Lee / Escribe el valor del temporizador para mantener una señal si no es su estado por defecto, por haber recibido un paquete de *I/O passing*.	0-0xFF (100 ms)	0xFF
P0	Selecciona / Lee la función a realizar por el pin PWM0.	0-2	1
P1	Selecciona / Lee la función a realizar por el pin PWM1.	0-2	0
M0	Lee / Escribe el valor de la salida de PWM0.	0-0x03FF	-
M1	Lee / Escribe el valor de la salida de PWM1.	0-0x03FF	-
PT	Lee / Escribe el valor del temporizador para mantener la salida de los PMW, cuando se recibe un paquete de *I/O Line passing* conteniendo información analógica.	0-0xFF (100ms)	0xFF
RP	Lee / Escribe el valor del temporizador para mantener la salida de PWM con el valor de RSSI del paquete recibido cuando está configurada para ello.	0-0xFF (100ms)	0x28
VR	Lee la versión de *firmware* del módulo.	0-0xFFFF	Prefijado
VL	Lee información detallada sobre la versión. Función obsoleta	-	-
HV	Lee la versión de *hardware* del módulo.	0-0xFFFF	Prefijado
DB	Lee el valor del RSSI (*Received Signal Strength*) del último paquete recibido correctamente.	0x17-0x5C	-
EC	Resetea / Lee la cuenta de errores ocurridos en el proceso de CCA (*Clear Channel Assessment*)	0-0xFFFF	-
EA	Resetea / Lee la cuenta de el número de errores, por no recibir la confirmación de recepción. (ACK, *Acknowledgment*)	0-0xFFFF	-
ED	Ejecuta un escaneo de la energía de los canales.	0-6	-
CT	Fija / Lee el valor del periodo de inactividad que estará el módulo en modo comando antes de salir de este.	2 – 0xFFFF (100ms)	0x64
CN	Hace al módulo salir del modo comando AT.	-	-
AC	Aplica los cambios de los parámetros que hayan sido encolados y reinicializa el módulo.	-	-
GT	Fija el tiempo de espera antes y después de introducir los caracteres para entrar a modo comando AT (GT + CC + GT), fijados por el parámetro CC.	2 – 0xCE4 (ms)	0x3E8
CC	Fija / Lee los caracteres ASCII, que se usarán para entrar al modo Comando AT.	0 - 0xFF	0x2B ('+')

Tabla IV.4: Tabla resumen de comandos AT